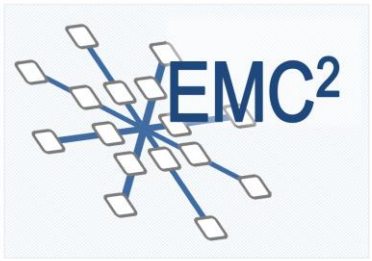

Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D1.5 – MW Framework Specification**	
Work package	WP1	SP_SoA - Embedded system architecture
Task / Use Case	T1.3	System service interoperability
Subtasks involved	ST1.3.3 MW Framework for System Service Interoperability	
Lead contractor	Infineon Technologies AG Dr. Werner Weber, mailto:werner.weber@infineon.com	
Deliverable responsible	Luleå University of Technology Jan van Deventer	
Version number	v1.1	
Date	07/04/2017	
Status	Final	
Dissemination level	Public (PU)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
16D	LTU	Jan van Deventer	Whole document

Document History

Version	Date	Author name	Reason
v0.1	27/04/2014	Alfred Hoess	Initial Template
V1.0	29/03/2017	Jan van Deventer	Meeting system requirements
V1.1	06/04/2017	Jan van Deventer	Final version
	07/04/2017	Alfred Hoess	Final editing and formatting; deliverable submission

Publishable Executive Summary

In the third amendment of the grant agreement, this deliverable D1.5 "*MW Framework Specification"* was combined with the deliverable D1.4 "*EMC SoA architecture and service proposals regarding inter-operability and service variability*" since the two documents overlapped considerably. They were both within the same task, T1.3 "*System service interoperability*". The Commission's automated administrative tool still expects a deliverable to be submitted. Duplicating the combined deliverable D1.4 "*MW framework Specification and service interoperability*" is not an option since it is confidential and the current one has a public dissemination level. A simplified version of the confidential document is presented here and is based on the final deliverable D1.2 of the work package, which proposes a reference architecture.

What we seek after is a middleware or framework for a service oriented architecture (SOA) where we can place applications (software modules) that are not frozen at design time and can handle changes in dynamic environments. The framework must insure cyber and IPR security. It must be dependable and allow for hard-real-time constraints with support of the other technical work packages of EMC². The framework must additionally promote interoperability between application.

The task was not to invent a new middleware, but look at the ones from previous projects ACROSS, INDEXSYS, GENESYS, MBAT, Socrades, IMC-AESOP, Arrowhead, ARAMiS, Euro-MILS, and SESAMO. It focused on the Arrowhead Framework due to its accessibility.

Table of contents

1. Introduction 6
   1.1 Objective and scope of the document 6
   1.2 Structure of the deliverable report 6
2. A Service Oriented Architecture 7
   2.1 The local cloud 7
   2.2 Core Services 8
   2.3 Support services 8
   2.4 Systems of Systems 9
3. Key themes 10
   3.1 Middleware Framework 10
   3.2 Interoperability and service variability 11
   3.3 Real time capabilities 12
   3.4 Security 14
   3.5 Dependability 14
   3.6 Runtime assessment 15
4. Conclusions 17
5. References 18
6. Abbreviations 19

List of figures

Figure 1: Local clouds in an MPSOC setting. 8
Figure 2: Mandatory core and Support Systems of the Framework. 9
Figure 3: A collection of applications with a minimum but mandatory middleware forming a local cloud. 10
Figure 4: Minimal local cloud with indications of mandatory core service interaction enabling service exchange between two application systems. 10
Figure 5: Simplified sequence diagram example of messages exchange in an Arrowhead Framework Local Cloud. 11
Figure 6: A System is capable of consuming the Framework mandatory core Services and will produce and/or consume one or more services. 11
Figure 7: The Translator system and its produced and consumed services. 12
Figure 8: Relationships between the different technologies within EMC2. 13
Figure 9: A two-way hardware authentication in conjunction with the core Authorization service. 14
Figure 10: The Quality of Service Manager and its produced and consumed services. 16

List of tables

Table 1: Abbreviations 19

1. Introduction

In the third amendment of the grant agreement, this deliverable D1.5 "*MW Framework Specification"* was combined with the deliverable D1.4 "*EMC SoA architecture and service proposals regarding inter-operability and service variability*" since the two documents overlapped considerably. They were both within the same task, T1.3 "*System service interoperability*". The Commission's automated administrative tool still expects a deliverable to be submitted. Duplicating the combined deliverable D1.4 "*MW framework Specification and service interoperability*" is not an option since it is confidential and the current one has a public dissemination level. A simplified version of the confidential document is presented here and is based on the final deliverable D1.2 of the work package, which proposes a reference architecture.

1.1 Objective and scope of the document

What we seek after is a middleware or framework for a service oriented architecture (SOA) where we can place applications (software modules) that are not frozen at design time and can handle changes in dynamic environments. The framework must insure cyber and IPR security. It must be dependable and allow for hard-real-time constraints with support of the other technical work packages of EMC². The framework must additionally promote interoperability between application.

The task was not to invent a new middleware, but look at the ones from previous projects ACROSS, INDEXSYS, GENESYS, MBAT, Socrades, IMC-AESOP, Arrowhead, ARAMiS, Euro-MILS, and SESAMO. It focused on the Arrowhead Framework due to its accessibility.

1.2 Structure of the deliverable report

This document contains two major sections. One considers the framework in general while the other looks deeper into the qualities expected by the EMC2 project.

2. A Service Oriented Architecture

Providing a Reference Service Oriented Architecture to be used on embedded systems that have multiple cores and are used in safety critical applications might seem initially puzzling. The applications do differ very much from each other, from automotive applications to avionics or even outer space ones. The embedded systems themselves also vary quite a bit, from heterogeneous dual cores to octo-cores and beyond on a single chip. But the reference architecture presented here depicts, at a concept level, how SOA can be implemented. To move from the abstract ideas towards concrete instances, the deliverable also points to examples developed in association with the Living Labs to how SOA has been actualized within EMC^2.

The concept of Service Oriented Architecture is not new. It has been part of the Internet especially with the World Wide Web. It relied on larger servers and much power. Moving the concept to constrained devices, i.e. embedded systems, is a natural evolution as the latter have become more powerful.

To avoid reinventing the wheel and leverage the results from previous projects, EMC^2 partners have used results from earlier projects. These include, among others, the GENESYS, INDEYXS, ACROSS, and Arrowhead projects (cf. D1.1 for their descriptions and relations to EMC^2). This technology has been used in different EMC^2 demonstrators. The Arrowhead project, like the EMC^2, has been a large ECSEL Joint Undertaking project. The project resulted in an open source SOA framework, which empowers interested parties to use the concepts with maximum advantage in their applications. The sheer size of the project implies an extensive set of documentation that is publicly available as it includes scientific publications [3], a book [7], and a wiki [2]. Moreover, the Arrowhead framework relies on numerous existing international standards, which promotes ease of systems integrations. Because of such reasons, the open source Arrowhead Framework describes well a general reference architecture for SOA towards applications with mixed criticality that use multi-core embedded systems.

The Arrowhead project's aim has been to enable collaborative automation by networked embedded devices. Its grand challenges were to enable the interoperability and integration of services provided by almost any device. This has been done by offering services established on the Internet Protocol Suite, which is a proven technology. The different software modules that are the service providers and consumers can be updated at anytime without affecting the other as they are loosely coupled and late binding. What is clearly defined are the interface themselves. Adhering to the international standards simplifies development and insures quality.

For low latency in control loops and increased security, the Arrowhead Framework proposes the idea of local clouds. Within such a local cloud, one finds an assortment of services, in the form of software modules, of which three are mandatory core services. The three mandatory core services are the Service Registry, Authorization and Orchestration. Of the many other support service modules, worth mentioning are the Historian, the Gate Keeper, the Quality of Service Manager, and the Translator [5]. The Historian is a database that can log events and signals, the Gate Keeper is the interface service in and out of the local cloud. The Translator is a service provider that intervenes transparently when different component suppliers have chosen different protocols, which could hinder collaboration due to protocol dialects [8]. The QoS manager keeps an eye on the running services to monitor the quality of service. The following subsections expand on these points and are further detailed in the middleware description.

2.1 The local cloud

The idea of the local cloud takes an interesting form in this project when being place in a multi-core embedded system context. Each core can hold its own cloud, or the multi-processor system on chip (MPSOC) can be a cloud on its own (cf. Figure 1). It is all up to the systems architects to decide what fits best for their applications.

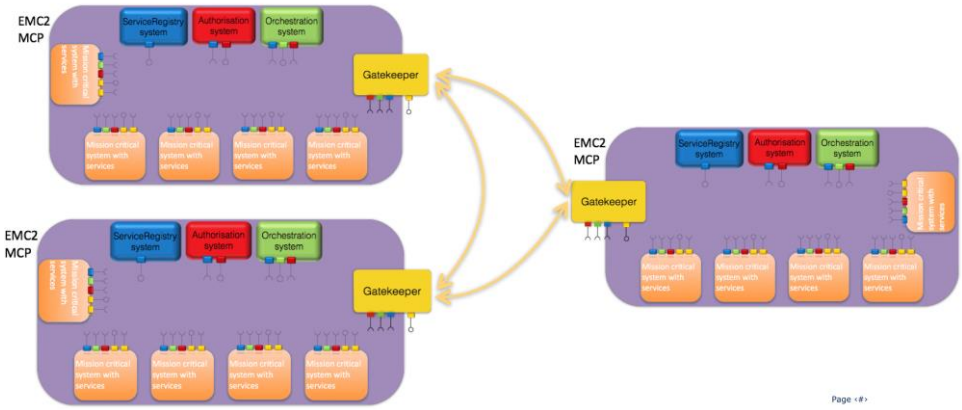

Figure 1: Local clouds in an MPSOC setting.

2.2 Core Services

Each cloud has its own set of mandatory core services necessary for the bare minimum SOA environment. This bare minimum has been set for very constrained systems. A collection of support services do exist and some are described in the next passage.

The core services are the Service Registry, the Authorization service and the Orchestration service. The Service Registry keeps track of all available services at all times. The Orchestration service provides the address of the most suitable services to a service consumer's request. The Authorization service ensures that the consumption of a service is authorized.

Another reason that these services are referred as core services is that they depend on each other since they interact with each other. For example, the Orchestration needs to check with the Service Registry to find out which are the current services available, and with the Authorization Service to ensure that the consumption of a service is allowed prior to suggesting it to the service consumer. When moving to more powerful processors, one finds quickly a need for additional support services.

2.3 Support services

The Arrowhead Framework offers a collection of support services (cf. Figure 2), which are well described in the wiki, book and publications. The deliverable D1.4 covers a more detailed and larger selection of support services. Four of these support services are introduced here as their existence addresses issues relevant to reference architecture and the requirements.

The Translator is a service that enables interoperability in a transparent manner. When a service provider registers its services, it includes details about the service, e.g., the address, port, service name, units, and protocol among other details. When the Orchestration finds the most suitable service provider but with some mismatch, e.g., protocol, it returns an address to service that belongs to the Translator. The Translator then intervenes in all communication between the service provider and consumer without either being aware of the issue. (This is part of the Interoperability presented later.)
The Historian is a database service that keeps track of information, e.g., signals. It can be queried and provides desired information in different formats to match the needs of the service consumer. Access to information from the Historian has to be authorized to insure information security. (This is part of the security presented later.)
The Quality of Service Manager is a service that keeps track of the quality of services and can flag undesirable or unacceptable behaviors. (This is part of the dependable SOA presented later.)
The Gate Keeper is the service one must go through to interact with the world outside the local cloud (cf. Figure 1). It therefore offers a security towards any threat from outside the local cloud. Inter cloud communication empowers the solution to form Systems of Systems.

2.4 Systems of Systems

The Arrowhead project did not invent anything revolutionary; it reused what was there. Service Oriented Architecture existed well before the project was initiated. What the Arrowhead project did was to build a framework that enable SOA within the world of cyber physical systems (CPS). Its aim from the start has been: technology to enable the interoperability and integrability of services provided by almost any device [4]. The project's vision was to promote collaborative automation by networked embedded devices.

Part of the success of the Arrowhead project is that this reuse of technology reckons on existing international standards. This translates into an ease of integrations of systems, which adopt the Arrowhead Framework, as SOA structure. The most prominent example of SOA, as an existing structure, is the World Wide Web that we surf with our web browsers. It is an impressive System of Systems example where components are dynamically changing. One of the questions at heart here is about its performance on multi-core processors within applications that have mixed criticality.

Through its dissemination effort, which includes its wiki, the Arrowhead project had developed a set of documentation and guidelines to enable system architects to leverage SOA with their needs to their advantage. For the next section, we address issues that were key to the EMC² project as it considered SOA in systems with mixed-criticality that dynamically adapt at runtime.

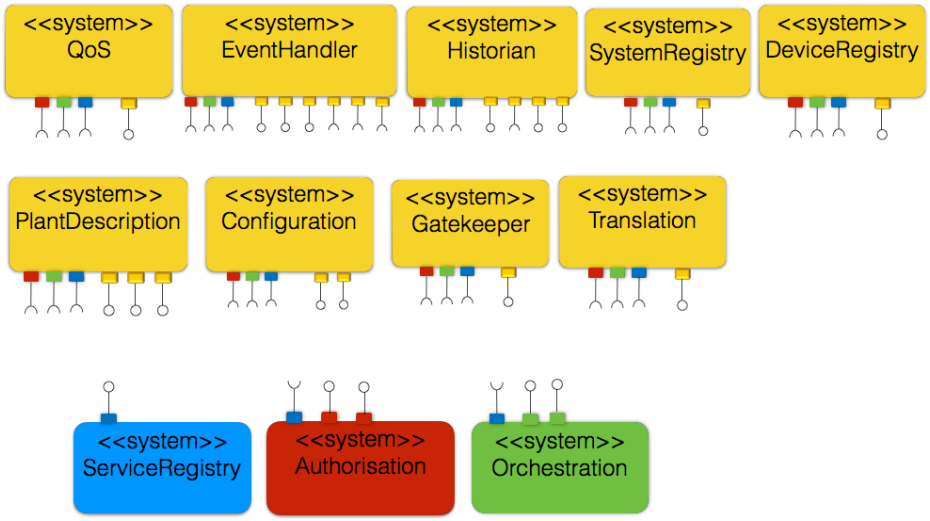

Figure 2: Mandatory core and Support Systems of the Framework.

3. Key themes

At the time of the project's conception, EMC² knew which would be key issues with the concepts of SOA in this specific context. It therefore broke down the SOA work package (WP1) in tasks that addressed those key issues. Each of these tasks had at least one deliverable that addressed the individual key issues. We therefore need to review them and insure that the proposed reference architecture is in line with each of characteristics reported in the deliverables.

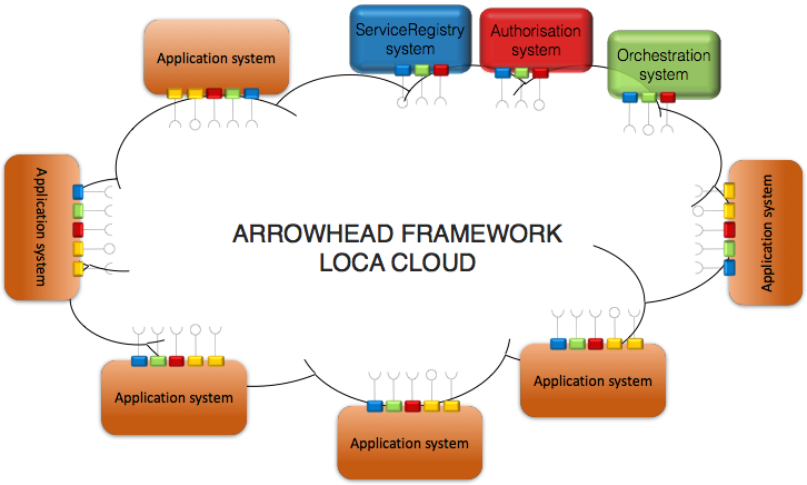

Figure 3: A collection of applications with a minimum but mandatory middleware forming a local cloud.

3.1 Middleware Framework

The term middleware refers to the software glue that connects the systems' applications in between themselves as well to the hardware interacting software. The reference architecture described earlier is this middleware framework that is the structure for the SOA concept. More details are available in the book and in the wiki. We briefly review it here for coherence through the herewith document.
In its bare minimum, a local cloud must be a self-contained and self-sufficient. It must contain the mandatory three core services and at least one application system. Figure 3 depicts such environment.

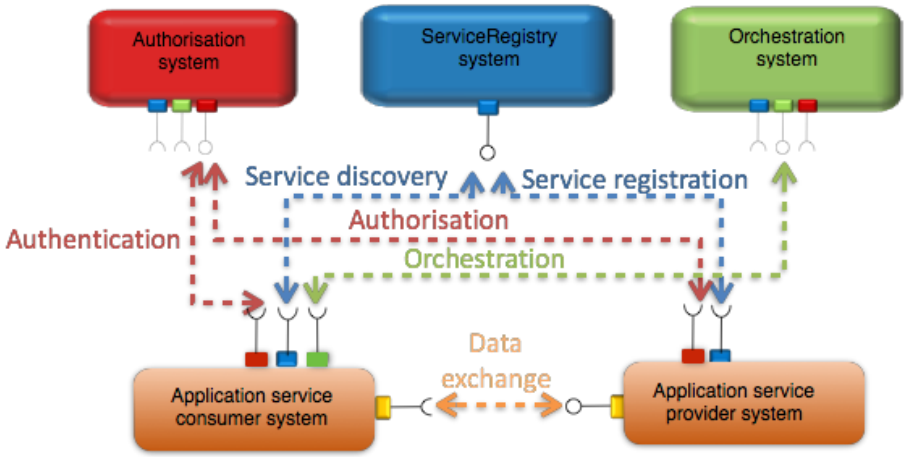

Figure 4: Minimal local cloud with indications of mandatory core service interaction enabling service exchange between two application systems.

The choreography of interactions between actors within the local cloud is described in Figure 4 and Figure 5. While registering its services, a provider must supply details about each service such as address, port, units, and communication protocol used, among others. This empowers the Orchestration to reference the

best service provider for the application at hand. For the purpose of making the figure readable, the sequence diagram (Figure 5) does not include repetitions or loops. In order to keep the Service Registry updated, the service provider refreshes the registration of its services. Similarly, the service request from the consumer to the provider is repeated at regular intervals or when needed (pull mechanism). For cyber and IPR security, the services have to be authorized with interaction with the Authorization service. The figure illustrates the case of a constrained device that uses a ticket. A ticket can be assigned a lifetime, which after expiration, needs to be renewed. The fetching of new tickets has an impact on execution time and power consumption (e.g., Volvo's wireless sensor nodes in their climate control demonstrator). The framework also support a push mechanism where the service provider pushes information upon events following an initial setup.

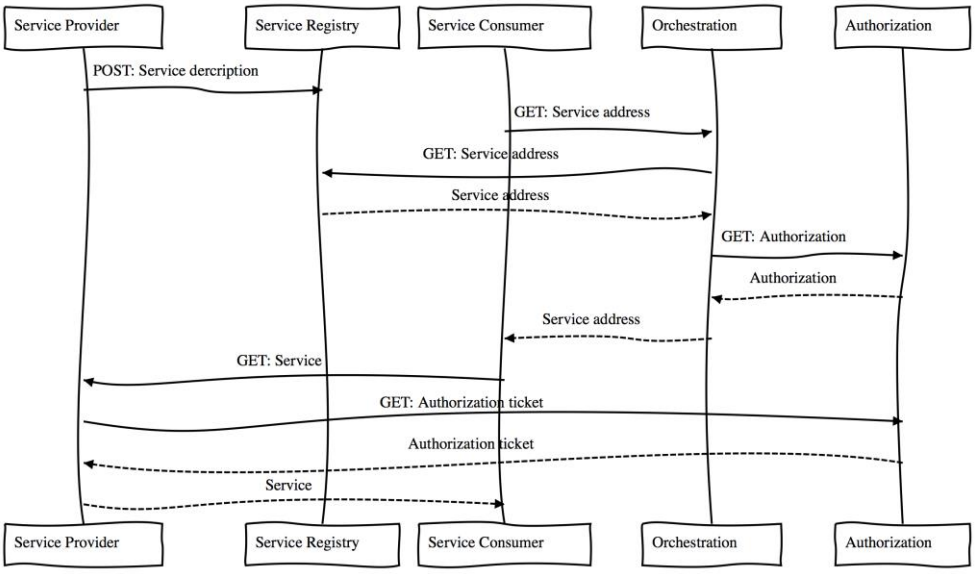

Figure 5: Simplified sequence diagram example of messages exchange in an Arrowhead Framework Local Cloud.

The Framework's documentation is detailed so that it is clear which services are produced and which are consumed by each of the components. This enables system developers to develop application system modules that easily integrate into local clouds. Figure 6 shows an abstraction of a system application module with its interfaces to its local cloud.

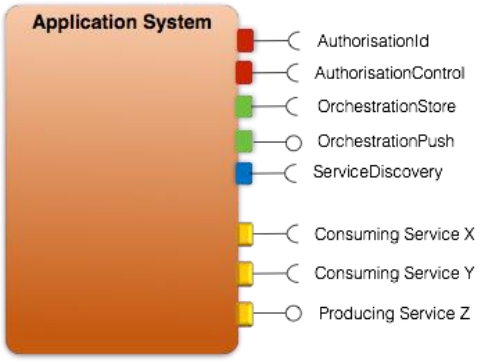

Figure 6: A System is capable of consuming the Framework mandatory core Services and will produce and/or consume one or more services.

3.2 Interoperability and service variability

In SOA, the interoperability between software components that offer services to the other components is through a communication protocol over a network. This idea can be condensed in information exchange.

The problem associated with the notion of interoperability in that context is the failure of communication. For example, on an airplane, a component from manufacturer A might use a proprietary protocol while another component from manufacturer B might not be using the same protocol; thus, this can lead to a communication failure. One could argue that the customer could or should impose the protocol but that is not the point here. The point is that things should just work together.

The Arrowhead Framework puts forwards a transparent Translator service to address the problem. It is invoked when a service consumer and a service provider do not "talk the same language".

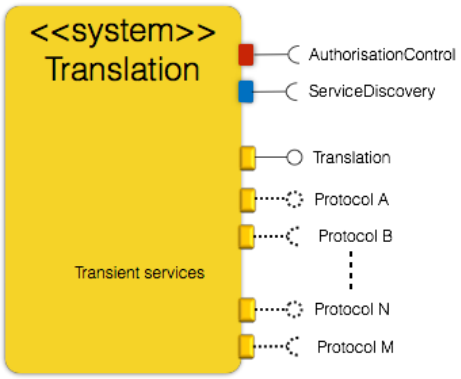

Figure 7: The Translator system and its produced and consumed services.

During the registry of a service, the service provider sends the name of the service, the address with port, among other descriptions. One of these descriptions is the protocol used by the provider. When a service consumer requests a service to the Orchestrator, it also states how it communicates. If there is a mismatch, the Orchestrator core service invokes the Translator service and provides details about the two stakeholders. To the service consumer, the Orchestrator returns the address and port to that specific translation service, without telling that it is a translation service. When the service consumer and the service provider communicate with each other, they are not aware of their "language" issues.

3.3 Real time capabilities

"Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments" is a challenging title for a project. Applications with mixed criticality imply that parts of the applications are safety critical. To some, that might point to real time capabilities when the system hard real time requirements. To the Living Labs in EMC², e.g., automotive, avionics or space applications, both topics are relevant. Deliverable D1.6, "System level convergence of real time capabilities", researched the topic. This has been divided into two parts: system level services supporting controlled mixed criticality operation, and performance predictability.

The deliverable refers to the safety related industrial standards (ISO26262, DO–178C and IEC61508) to discuss the topics at hand. It has an interesting section (§2.3) warning about the differences between theoretical models and industrial practices. They include misalignment of terminology, software task assurance level and the notion of importance, different worst case execution time estimates, and graceful degradation. This points to the importance of EMC² where different communities have to interact with each other, and these misalignments become visible. This is further enhanced when considering the real-time aspect of the reference SOA.

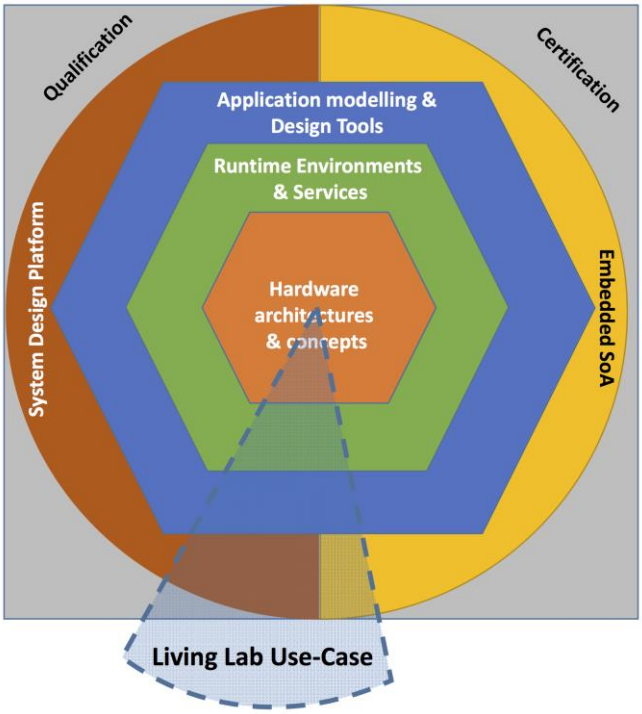

Figure 8: Relationships between the different technologies within EMC2.

The Arrowhead Framework recognizes the vital role of real time aspect of the system. Its book states: "meeting such hard-real-time requirements is critical to production system performance." However, the framework cannot by itself guaranty hard real time because SOA sits on top of some operative system and/or hypervisor running on some multi-core processor. It would have a hard time to impose real time requirements. To achieve this, different technologies have to work together as they do in the EMC^2 project. Figure 8 shows how the different technologies (WP 1 – WP 6) relate to each other and how the Living Labs integrates these technologies.

In D1.6 §4, AICAS clearly proposes a possible solution to this problem, which is to use hardware enforced time and space isolation. With this idea, applications run in their own memory partition and their own time slice. The bounds of the memory partition are defined. The time slices are scheduled based on application importance and on analysis combining all those different factors that defines clear rules to be respected during the development process. A time slicing example in EMC^2 is the Offis copter demo that prioritizes the flight control over the video image processing when necessary (WP2).

The concept of performance predictability in dynamic and changeable real-time environments might sound like an oxymoron, especially when reflecting on the sequence diagram of Figure 5. There are a lot of messaging going on; an authorization ticket might have expired or some services might have drop out, requiring a new one to be orchestrated and authorized. But proper analysis such as worst case execution time will provide indications of the performance. Within EMC^2, the University of Manchester has been developing a tool to support the analysis. On the other hand, at runtime, the Arrowhead Framework puts forward its Quality of Service Manager to monitor or assess the performance and potentially make changes when performance does not meet the requirements (cf. § 3.6).

Multi-core processors additionally offer system architects the option to pick and choose. As Volvo suggested in D1.6 §4 that safety critical components can be separated from infotainment systems that use SOA. The example exhibiting this idea is the model car with its heterogeneous dual core. On the Cortex M4 core, a real-time OS handles all the safety critical aspects of the car, e.g., drive by wire, ABS, and traction control. On the Cortex A9 core, the Arrowhead Framework, on a Linux OS, logs the vehicle's states and signals with its Historian (database) service as well as making them available as a service on Ethernet (and WiFi) over the CoAP and HTTP (transparent translation) protocols.

Integral multi-core and mixed criticality-aware designs can also be leveraged on the increased tool support from EMC2 results. Variability management, dynamic re-configuration and adaptivity support can work together to enable multi-criteria optimization that still ensures proper real-time support. Proof of this is the implementation of multi-core variability management designs for access control and situational awareness presented in WP12.2(c) (see D12.5). There, the measured performance is fed back into the design so that an adaptivity management library (*pappadapt*) chooses between differing OpenMP implementations of video analytics. This design effectively integrates a big portion of the overall picture in Figure 10 for design through runtime.

3.4 Security

Information security has become a very essential topic, and especially in safety critical systems. News reports of relevance in EMC² gives examples of cars connected to the Internet being hacked in and their control taken over. Security is an integral part of the Arrowhead Framework. Among the mandatory core services is the Authorization service.

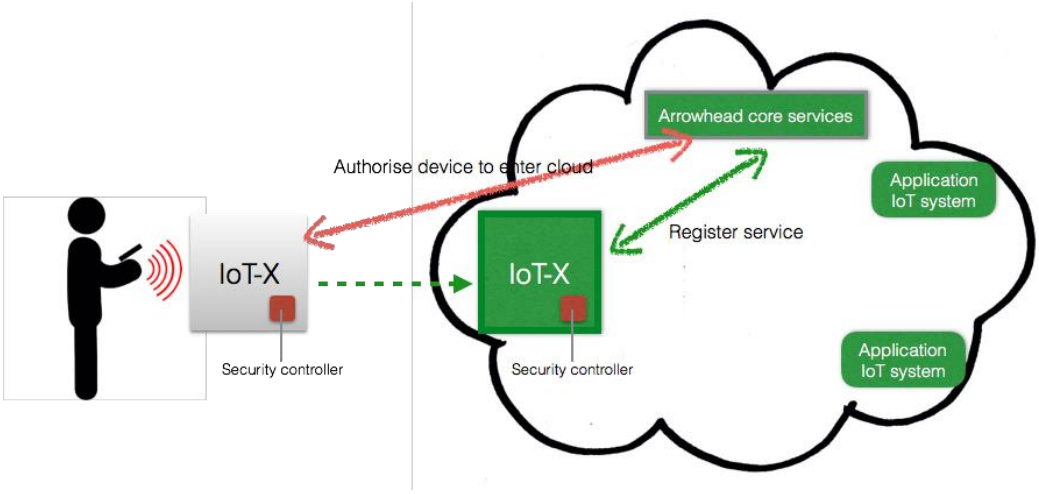

Figure 9: A two-way hardware authentication in conjunction with the core Authorization service.

Deliverable D1.7, Security services of the EMC² architecture, addresses the topic. Among the mechanisms presented, are tickets (e.g., Radius or Kerberos) [6][14] and certificates (e.g., X.509) [13]. Tickets are used with constrained devices (e.g., I/O nodes), while certificates are more appropriate with more powerful processors.

Some of the project partners were both in the EMC² and Arrowhead projects (e.g., LTU and Infineon) and influenced each other. This has led to an interesting use case and automotive demonstrator [12]. A system supplier on a vehicle can access information only about its own system onboard as authorized by the vehicle manufacturer and the system (software level agreement, SLA). The demonstrator uses hardware authentication to guaranty the identity of the components, i.e. it cannot be faked or disguised. The framework safeguards the IPR of the information. An interesting extension to this security scheme is that intelligent I/O nodes (e.g., sensor and/or actuator nodes) can have also such hardware authentication to ensure security as far as possible. Such security scheme can include humans (cf. Figure 9), system installation, deployment, configuration, and reconfiguration.

3.5 Dependability

In the context of SOA, dependability can be seen as a property of a system that provides services, which are developed with respect to dependability attributes and means to react on risks or threats. As part of the task “Safety and Fault-tolerance Concept” (T1.6), the concept of system (and its components) dependability

was considered. The dependability of a system is its ability to deliver specified services to end-users so that they can justifiably rely on and trust the services provided by the system [11]. Dependability includes several aspects of a system. All aspects are intensely domain, target and application specific. Therefore, with respect to mixed-criticality and multi-core, D1.8 defined the following attributes for a dependable SOA (dSOA): Attributes are:

- Reliability - continuity of correct service,
- Safety - absence of catastrophic consequences for user(s) and environment,
- Security - protection against malicious user(s) and misapplication (cf. § 3.4),
- Adaptability - readiness for upgrade and update,
- Reusability - readiness to use the service in different systems, devices,
- Availability - readiness for correct service,
- Maintainability - readiness for modification and repairs,
- Integrity - absence of improper system alterations,
- Confidentiality - readiness for Trusted Computing (cf. § 3.4).

To design a dSOA, the whole Development Life Cycle (DLC) has to align to an adequate standard. The ISO 26262 [10] and the IEC 61508 [9], which are generally used in automotive and industrial domains, are not sufficient, because they require that all components already are known by the development phase. These dependable components must be able to register their services at runtime to integrate into SOA.

In a context with mixed criticality, it is essential that one can depend on the information received from the different services. The trustworthiness and origin of service providers and consumers are insured by the security mechanisms (authentication and authorization). Nonetheless, things can go wrong. e.g., a temperature sensor might fail or the magnet of a reluctance sensor might be contaminated with iron fillings resulting in an intermittent multi pole sensor. When an anomaly is detected, the service consumer is redirected to the best available service, if such is available. Virtual Vehicle demonstrated the concept at the second review of EMC^2 in Gothenburg using an Aurix multi core chip with ROS (Robotic Operating System). There a sensor failure was artificially induced, it was detected, and the SOA system recovered from it.

The proposed reference architecture promotes dependable SOA with support services such as the EventHandler, Quality of Service Manager, and Application Manager. In the event of any anomaly, the EventHandler system (c.f. Figure 2) will notify the Orchestration and the Quality of Service Manager. This can be the case of a service detecting aquaplaning. There is also a clear connection between the anomaly detection mechanisms and the Orchestration. Discovering an irregularity based on the available information uses a similar logic as selecting the next best service provider. For example, on an active safety system, an abnormal wheel speed can be detected from the opposite wheel and output shaft angular speed sensor when an open differential is involved. The next best service might just be a calculated wheel speed from the other two sources. That is, dependable services can not only verify the proper behavior of other services, but can offer themselves as a redundant back up service until the system is mended. In section 3.6, which considers runtime assessment, the Quality of Service Manager, and Application Manager are introduced. Together, the core and support services address the above list of dSOA attributes.

3.6 Runtime assessment

A partial goal of the project has been to evaluate applications mixed criticality in dynamic and changeable real-time environments using SOA. The interest in this research question is fueled by the desire of additional flexibility such that a system is not fixed or locked at design time. The flip side of this desire is the fear that when things change at runtime, bad things happen in applications that are safety critical.

Deliverable D1.9 “Runtime assessment architecture support concept: design principles and guidelines” reviewed the means to achieve that. The question here becomes: how the proposed reference architecture address and handles the issue? Building a system that must adapt during execution time based on available services requires continuous assessment of the services and system’s performances. For this, the Arrowhead

Framework relies on the support service named Quality of Services (QoS) Manager. The QoS Manager system's objective is to verify, manage, and guarantee QoS for services.

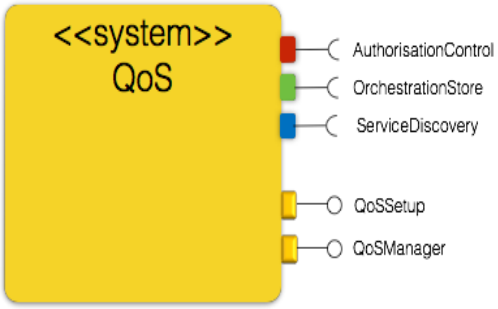

Figure 10: The Quality of Service Manager and its produced and consumed services.

Quality of Service (QoS) within a local cloud is important. The requirements on mixed criticality applications related to real-time communication and security have to be fulfilled. To achieve them, both monitoring of QoS and mitigation of QoS deviations are supported within a local cloud. In the Arrowhead Framework architecture, the QoSManager system [1] supports QoS configuration and monitoring, in close collaboration with the Orchestration system.

Most of Arrowhead matchmaking between service producers and consumers are driven in a declarative manner: the Orchestration system interacts with DeviceRegistry, SystemRegistry, ServiceRegistry and PlantDescription systems to produce orchestration rules to individual application systems. Such orchestration data must consider the QoS requirements set by individual application systems. These QoS requirements are considered as constraints on the matchmaking. The QoSSetup service acts as a support service to the Orchestration system. For every change in a local cloud, the resulting QoS has to be predicted. The changes cause the Orchestration system to compute alternative orchestrations, which should be verified through the QoSSetup service. This will be repeated until a specific set of orchestrations appears to support the required QoS. Once the orchestration is settled the Orchestration system requests the QoSSetup service to perform the reservations necessary to grant the QoS. The Orchestration system also distributes the service end points to the systems involved.

Additionally, in its reference implementation, the Arrowhead Framework has an Application Manager. This Application Manager can terminate, start or restart any system application that behaved inconsistently. This has not been evaluated within EMC[2]. The thought has been to evaluate the concept using containers, but due to time limitations, it has not been tested.

4. Conclusions

This deliverable presents a Service Oriented Architecture middleware or framework. It addresses issues such as lowering the cost of software development and maintenance by having application or software modules that are not set at design time but can evolve with time. This is because they are loosely coupled and late binding. To make this possible, a registry of services offered by these applications must be maintained up to date at runtime (it is done by the Service Registry). The coordination between service providers and service consumers is done by the Orchestration service. The Authorization service ensures that these service exchanges are permitted. There is a collection of other support services to enable more complex situations. One of them it the Translator, which is invoked by the Orchestration service when there is a communication mismatch between the service consumer and the service provider.

5. References

[1] Michele Albano, Ricardo Garibay-Martinez, and Luis Lino Ferreira. Architecture to support quality of service in arrowhead systems. In Proceedings of INFORUM 2015, Covilhã, Portugal, Sep. 2015

[2] Artemis project Arrowhead. *Arrowhead Framework Wiki*. https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Arrowhead_Framework_Wiki, accessed 2017-03-09.

[3] Artemis project Arrowhead. *Arrowhead Publication Materials*. http://www.arrowhead.eu/material/, accessed 2017-03-09.

[4] Artemis project Arrowhead. *General Overview*. http://www.arrowhead.eu/about/general-overview/, accessed 2017-03-09

[5] Artemis project Arrowhead. *Support Core Systems and Services*. https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Support_Core_Systems_and_Services , accessed 2017-03-09.

[6] A DeKok and A Lior, *Remote authentication dial in user service (radius) protocol extensions*. Technical report, RFC 6929, April, 2013.

[7] Jerker Delsing, editor. *IoT Automation: Arrowhead Framework*. CRC Press, 2017.

[8] Hasan Derhamy. *Towards Interoperable Industrial Internet of Things : An On-Demand Multi-Protocol Translator Service*. PhD thesis, Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering.

[9] *Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 2*: *Requirements for electrical/electronic/programmable electronic safety-related systems*. Standard, International Electrotechnical Commission, Geneva, CH, March 2010.

[10] Road vehicles: *Functional safety, Part 1: Vocabulary*. Standard, International Organization for Standardization, Geneva, CH, November 2011.

[11] Jean-Claude Laprie. *Dependable computing: concepts, limits, challenges*. In In Proceedings 25th IEEE International Symposium on Fault-Tolerant Computing. Citeseer, 1995

[12] Christian Lesjak, Daniel Hein, Michael Hofmann, Martin Maritsch, Andreas Aldrian, Peter Priller, Thomas Ebner, Thomas Ruprechter, and Günther Pregartner. *Securing smart maintenance services: Hardware-security and TLS for MQTT*. In 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pages 1243-1250, July 2015.

[13] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. *X. 509 internet public key infrastructure online certificate status protocol-OCSP*. Technical report, RFC 2560, 1999.

[14] B. C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. IEEE Communications Magazine, 32(9):33-38, Sept 1994.

6. Abbreviations

Table 1: Abbreviations

Abbreviation	Meaning
CPS	Cyber Physical System
dSOA	Dependable Service Oriented Architecture
DNS	Domain Name System (server)
EMC²	Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments
IoT	Internet of Things
µC	Micro-Controller
SLA	Service Level Agreement
SOA	Service Oriented Architecture
URI	Uniform Resource Identifier
WP	Work Package