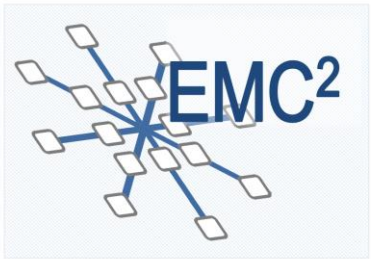

Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D1.9 – Runtime assessment architecture support concept: design principles and guidelines**	
Work package	WP1	SP_SOA - Embedded system architecture
Task / Use Case	T1.6	T1.6 Safety and Fault tolerant concepts for SOA Embedded system Architectures
Subtasks involved	ST1.6.1 Safety concepts on architectural levels	
Lead contractor	Infineon Technologies AG Dr. Werner Weber, mailto:werner.weber@infineon.com	
Deliverable responsible	Alten Sverige AB Detlef Scholle, detlef.scholle@alten.se	
Version number	v1.0	
Date	16/05/2016	
Status	Final	
Dissemination level	PU	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
16M	ALTEN	Detlef Scholle Jens Berman	Basis for the delivery
01B	AICAS	James Hunt	Extended Security section

Document History

Version	Date	Author name	Reason
v0.1	22/02/2016	Detlef Scholle	Initial draft
v0.2	23/04/2016	Detlef Scholle	Improved for internal review
v0.3	02/05/2016	Detlef Scholle	Improved after internal review
v0.4	03/05/2016	James Hunt	Improved security discussion
v0.5	13/05/2016	Detlef Scholle	Improved after formal internal review by Volvo
v1.0	16/05/2016	Alfred Hoess	Final editing and formatting; deliverable submission

Publishable Executive Summary

The abstract of application usability may be described as services. Services are loosely coupled functions, tied together. In the interconnected world of systems this is already the fact for cloud base systems. For embedded systems on other hand, this has instead promoted a different deployment model. The traditional embedded solution was one feature provided as one isolated stand-alone system. Unfortunately, this is not acceptable for several reasons (energy consumption, costs complexity, etc.) For modern embedded system the scope is to deploy features as a combination of services and the collaborating environment is a local cloud (or fog).

This is enabled via the design pattern Service Oriented Architectures (SOAs). Classical SOA is not acceptable for integrating embedded multi-core devices in a safe and secure manner since it lacks characteristics in context of dependability requirements demanded by embedded critical applications. The dependable Service Oriented Architecture (dSOA) is the intended solution in this delivery. Section 2 describes the scope of work.

Table of contents

1. Introduction 6
1.1 Objective and scope of the document 6
1.2 Structure of the deliverable report 6
2. Safe and Dependable Service Architecture 6
2.1 Defining scope to target, functionality for SOA 6
2.2 Mixed-Criticality 7
2.3 Run time design principles of safety SOA 7
2.3.1 Services with a common data model 7
2.3.2 Services with a common message framework 8
2.3.3 Services with a service contract 8
2.3.4 Services locating service contracts 8
2.3.5 Services support on security 8
2.3.5.1 Identification 9
2.3.5.2 Authorization 9
2.3.5.3 Permissions 10
2.3.5.4 Monitoring 10
2.3.5.5 Enforcement 10
2.3.5.6 Portability 10
2.3.6 Services support on reliability 10
2.3.7 Realtime embedded systems 11
2.4 Binding services and its protocol 11
2.4.1 Safety services validation 12
2.4.2 Dynamic reconfiguration 12
2.5 Local cloud 12
2.5.1 Resource Deployment Management Service 12
2.5.2 SOA for mixed criticality and Extended ECU 13
2.6 Requirement coverage 14
3. Conclusions 15
4. References 16
5. Abbreviations 21

List of figures

Figure 1: General view of mixed criticality ECU 14

List of tables

Table 1: Abbreviations 21

1. Introduction

1.1 Objective and scope of the document

This formal delivery document should report the design principles and guidelines for architecture support for runtime assessments. It is the result from the elaborations within T1.6 concerning safety and fault tolerant concepts for runtime support for Service Oriented Architecture (SOA) of embedded systems.
According to the DoW the EMC2 project will in Task 1.6 focus on Safety and Fault tolerant concepts for SOA Embedded system Architectures. This also involves the specific work related to ST1.6.1 on safety concepts on architectural levels.

One fundamental view of this topic is the Service Oriented Architectures, SOA. To enable Safety involves various concerns. An additional concern is the mixed criticality. The system will hereby, support both safety components and non-safety components. This will affect system operation management, maintenance and diagnosis services enabling fault tolerant system operation thus supporting system properties like e.g. availability, robustness, reliability, fail safe operation, dependability. One main target of this task is already mentioned, the special challenges regarding functional safety in EMC2 systems and SOA.

For traditional systems, applications are often statically scheduled as if they would be located on a single core. The aim in EMC2 is to develop a more dynamic approach, which are not supported or even prohibited by current safety standards.

The project's assertion that multi-core systems pose special requirements on certification guides the task. One main challenge is to show the independence of elements required for ASIL decomposition as proposed for example in ISO 26262 for EMC2 systems with shared memory spaces and for mixed criticality. In general sharing the volume (typical time or location) is managed by dedicated volume for selected safety component. Hereby safe partition is established.

From the view of SOA this cover issues related with dynamic reconfiguration, and also the actual existence of multiple cores. It will include the application of cooperating task in order to evaluate with respect to the safety of a system.

1.2 Structure of the deliverable report

This delivery is reporting the subtask SP1.6.1 of T1.6 Safety concepts on architectural levels. The report starts with a walk through section 2 with an overview on SOA toward EMC2. Here with the scope on the automotive domain in general, and involving the concept of dependable Service oriented Architecture (dSOA). The section is concluded in section 3. Section 4 provides links with references.

2. Safe and Dependable Service Architecture

2.1 Defining scope to target, functionality for SOA

A SOA is abstracting applications by defining "functional snippets" that will provide useful value. This useful value is generalized in the sense that is it might provide more flexible usage or multi-purpose usability. A set of services is possible to be combined to define a specific "application". In case of static configuration, the set of services are bound statically before online. In cases with autonomy the services are instead able to be bound on the fly and later reconfigured if needed. A reconfiguration may be to handle a malfunction or to perform an optimization of the system.

In the safety case, the system that is certified requires to be isolated from any type of external impact. This system is then the object for verification.

The runtime guidelines are required to include both temporal as spatial portioning. The motivation for this is according to mixed criticality. Mixed criticality services that are not correctly designed with good partition may interfere with other safety-critical services. When services share the same resources, unpredictable things may happen. Services of lower criticality can, for example, overwrite memory. If a service overwrites data of another service, the other service might behave unpredictably. Spatial partitioning [53], i.e. encapsulation/integrity of data, can avoid these types of issues, and is important to avoid this type of failures to propagate to safety-critical services.

2.2 Mixed-Criticality

Mixed-criticality system is able to execute both safety and non-safety software components. Both types are executed on the same hardware platform without risk for interference. To co-exist they must not affect the isolation or the partitioning of the two critical instances. A safe system is definitely deterministic but a real time system might not be safety.

Executing software that is non-critical in the same system where safety software is executed requires fulfilment of specific requirements. Any resource that are representing sensors, actuators, time, memory, storage, cores, etc. are accessed by both non-safety and safety tasks. The specific requirement is that any task that is safe shall be isolated and partitioned to ensure any risk for interference. The scope in this report is how services based systems may manage the mixed criticality criteria. The requirement is then to protect the operation of one critical component from any kind of faults and from any other critical level component. It is significant to handle components differently when they are defined with different criticality levels. According to Alan Burns et. Al [52], such protection of all components need to be engineered to the strict standards.

The core rule with safety-critical system is to have a common strategy for approaching safety lifecycle activities like initial concept, design, implementation, operation and maintenance.

The safety standards are required to be considered as integrated in the design architecture of the SOA. As an example the ISO 26262 divides the functional safety of the road vehicles into four safety levels, aka Automotive Safety Integrity Levels (ASILs). The ASILs considers the level of severity, probability of exposure, and controllability. The Severity is broken into four levels ranging from no injuries to life-threatening injuries. Probability of exposure (i.e. failure) is broken into five levels ranging from *incredible* (very unlikely) to high probability. Controllability by the driver is broken into four levels, ranging from *Controllable in general* to *Difficult to control or uncontrollable*.

The consequence is that the architecture must provide a framework for managing the various level of safety, as it will transport and manage all kind of software (code and data).

2.3 Run time design principles of safety SOA

This section is defining the general description of the services required. Safety cannot be ensured unless security and reliability is ensured. This section describes how these properties are addressed in SOA-standards.

2.3.1 Services with a common data model

The messaging model refers to the mechanism getting the services to interact with each other through messages. Basically this is defining that managing services include parsing, generating and validating data. The criterion involves the interaction pattern, Quality of Service (QoS) and data format. QoS concerns both security and reliable management depending on criticality level. Simplifying the data model, the services require having a common data format and to use parse/generate tools and validation

tools. A consequence is then one tool and structure for all the communication. A level of flexibility is needed since values, including parameters and variable data, seldom are static. They change over time. Data is additionally validated according to schemes for two reasons; first, a service consumer may use to generate a message that a service provider understand in terms of structure. Also verifies that the received data is of correct data types. Secondly, the service provider may validate that the return messages received is valid to provide safety against bogus values.

Examples of protocols that provide these features are SOAP or REST WebService. Communication is not always done through one-way transmissions, and there are a couple of common interaction patterns. Request/reply interaction is the pattern where the service requester sends a request and waits for a reply. Request/callback interaction is similar, but the requester does not wait for a response, instead is called on when the response arrives. Asynchronous Store-and-Forward messaging is an architecture where the requester puts the request in a persistent queue. This allows the requester to continue working while the queue reliable delivers the message. The publish/subscribe paradigm enables services to subscribe on event types, and receive notifications when a service generates those events.

2.3.2 Services with a common message framework

A general framework for common service management is required. Any services need support to send relevant messages to any other service. Messages also include non-functional properties or elements typical security tokens, timing constraints etc. Side note, it might be good to be able to specify intermediary notes.

The data model is important so that services understand each other. The service provider needs it to accept and generate data values according to its service contract. The service consumer needs it to send and receive the correct data type. This means that SOA needs a way of parse, generate and validate data. By using a common data format, one type of tool is needed instead of using different tools for different services. The validation checks which elements and attributes are allowed in the document, and what order and relation between the elements that are allowed. By using the validating document, parsers can recognize errors, and data generators have knowledge of how data should be generated. If the data format is not common on the other hand, developers need to extract/generate the information in specific ways for each service they create. This is bad for composite services that might need to create several tools.

2.3.3 Services with a service contract

This enables services to find a service based on what it provides. It tells the consumer how to use it, if it can use it and where it can be found. Examples of contracts are WSDL (in relation with SOAP) or WADL (in relation with REST) or IDL. A service should not only be able to locate run-time service instances, but also manage storage and look up service contracts. Furthermore, adding taxonomies so that services can be categorized to enable better searching capabilities (yellow-page structured register).

2.3.4 Services locating service contracts

This means that there must be a common way to find service contracts (not just run-time service instances). This could be anything as simple as broadcasting them (very static) to UDDI (very dynamic). There is a trade-off between simplicity and sophistication. Examples of lookup services are UDDI or Web Services (CoRE link). Choosing the service contract register consider a trade-off between simplicity and sophistication. A pragmatic and also a very static approach, is applied by letting the service provider send the contract to the service consumer directly. An additional dynamical approach is to use a WSDL-repository. Besides being a repository of WSDL documents, it may provide service notifications and better searching techniques. UDDI is an increased level more sophisticated and is hereby dynamic but instead lacking the simplicity.

2.3.5 Services support on security

This is an indirect topic for this deliverable. It is concerned as a design requirement. Security includes the properties of confidentiality, integrity, and authentication. Privacy and confidentiality implies non-

interception of communication including reading of the actual data, and ensuring the encryption. Integrity combines the encryption and the signature mechanisms to protect from data that is altered. Authentication confirms the identification of the service, and hereby prevents malicious services that unauthorized might harm the system. Several mechanisms are available to ensure security properties but not always usable in resource constrained components or even more likely unsuitable in embedded real-time systems. Wireless network communication is definitely a specific topic not in scope for this deliverable. Wireless network communication increases the risk for attacks to consider, e.g. jamming attacks where attackers drops the signal strength.

The major issue for dynamic systems is security. The main questions are;

- what code,
- from whom,
- may run on what system,
- when,
- with what resources and
- with what access to the system?

It is up to the runtime system to ensure that however these questions are answered, the runtime can ensure that system is only used in accordance to those answers. There are five major aspects that are important for the runtime assessment: identification, authorization, permissions, monitoring, and enforcement. These can be addressed in turn.

2.3.5.1 Identification

Identification has two sides, the source of code and data to be used on the system and the system itself. In general, identity certificates can be used to determine who is who. The X.509 encryption standard provides an adequate means of creating and managing such certificates. A certificate authority provide a positive means of identifying to whom a certificate belongs.

A certificate can be used to demonstrate that some information has been provided by the owner of the certificate. Each certificate carries public key within, which can be used to decode information provided by the owner of the certificate that has been encrypted with the owner's private key. The central problem is to ensure that no one other than the certificate owner can access the private key.

For identifying an organization, this is not so difficult; but for a device, this is more of a problem. An organization can keep a private certificate under lock and key. It generally has full control over its servers. Devices on the other hand are sold and distributed. It is much more likely that a device might be opened and the private key duplicated to another device. Some hardware-based mechanism is needed to shield private keys from prying eyes.

Being able to identify both organizations and devices, one has a means of distinguishing from whom code can come and where it runs, but this is just the beginning. Identification verification needs to happen whenever connections are made between devices or devices and servers and when code is run on the system. Once identified, authorization can take place.

2.3.5.2 Authorization

Authorization is needed for validating services, devices and code. After identifying the source or destination of a message, one can decide whether or not to act upon the message. Likewise, once one knows from where code comes one can decide whether that code may run on the current device or not. This entails checking signatures against recognized certificate authorities. This can be done with direct authorization, i.e., the system can recognize a key directly as being authorized, or one has a chain of trust from some source of authorization. In the second case, one may recognize any signature that has been signed by a known authorization agent. Again the X.509 standard provides means for doing this.

2.3.5.3 Permissions

Permissions are the next step. The system requires both a means of associating a set of permissions with each certificate and a means of ensuring that code signed with a given certificate cannot use any API for which it does not have the requisite permissions. A single example of such a system is the mapping between users and files system and execution permissions provided by many operating systems. This is a relatively coarse grain system that can only provide limitations at process boundaries. SELinux provides a bit more enforcement on top of the base Linux capabilities, but this is still process based. The Java security model provides a much finer grained differentiation that works with a process. The main point is to have such a system that meets the security requirements of the system. The more general the system definition the more encompassing the system needs to be.

2.3.5.4 Monitoring

No system is completely immune to penetration. System monitoring should be used to ensure that only the expected processes run on the system and they remain responsive. Of course, in a dynamic system, such expectations must be managed. As the application palate changes, the system needs to update its list of expected applications and behaviour. For this, a means of validating configurations is necessary as well.

2.3.5.5 Enforcement

Permission checking can be used to deny access, but not to limit rates. This is particularly true of CPU use. Here, it is not enough to monitor how much each CPU time each process or task is using, but also to ensure that when given limits are reach, that the application in question can be throttled back.

In general, a thread can either be running or not. Any rate limitation has to be based on some period of measurement. For example, if a thread should not be able to use more than 10% of the CPU, this has to be translated into a fraction of the measurement period. The short the period, the more enforcement costs, but also the more responsive the system is. For realtime tasks, the enforcement should be based on the tasks deadline. For other tasks, it is less critical.

Once a given CPU use, often referred to as cost, has been reached within a given period, one would like to stop the thread until the next period. Though this can be done for full processes, it is dangerous in general. Just stopping a thread that holds a lock can cause deadlock, therefore it is better to reduce its priority to below all other threads in the system. Then it can continue to make progress, but more importantly, it can continue to take part in priority inversion avoidance mechanisms, such a priority inheritance. This allows resource to be timely released even in enforcement mode.

2.3.5.6 Portability

These basic mechanisms need to be supplied by the platform for dynamic multi-critical systems to be viable. There are many ways for these services to be provided, but they become more attractive with a standardized runtime environment. This means either standardizing on one CPU architecture and operating system or employing virtual machine technology. A Java byte-code based infrastructure, with deterministic garbage collection, could provide these services independent of the underlying OS and processor, especially since OSGi provides much of the necessary framework for dynamic code replacement.

2.3.6 **Services support on reliability**

This is an indirect topic for this deliverable. It is concerned as a design requirement but not covered in this text. In the case of mixed-critical systems, the scope is to consider fault handling to avoid unwanted

consequences on unexpected events. Here is the isolation part of critical partitions relevant. The critical part requires protecting (and part of certified) from a gate keeper preventing (stopping from propagating) external fault injection.

Reliability in SOA is typically concerning the reliability of messaging. Reliable messaging include that messages that are sent are guaranteed to be delivered without duplicates and in the same message order. This is typically provided by TCP on the delivery channel, however that does not guarantee these properties for the end-user.

To provide reliable messaging to the end-user messages can have message IDs, sequence numbers, and acknowledgements, it should also be able to be retransmitted if necessary [1]. A service initiator starts by sending a sequence of messages to another service. The other service acknowledge which messages it have received. If the initiator sees in the acknowledgement that a transmitted message haven't been received, the initiator retransmit that message only.

The different type of delivery guarantees are such as *at most once*, *at least once*, *exactly once*, and *in order*. Both the *at most once* and the *in order* is to guarantees duplicates. Both the *at least once* and the *exactly once* guarantees delivery of the message. The *in order* means that the messages are received in the same sequence order as sent.

2.3.7 **Realtime embedded systems**

The SOA may require managing the whole range from realtime to non-realtime systems. The static planned scheduling approach for critical tasks is a pragmatic angle for certification (resources as time allocations for task are pre-determined). The cyclic approach uses a periodic task model (any task type are handled and periodically executed). Harmonic task periods are suitable to simplify the complexity in both time and space. Achieve by temporal isolation requiring task with lower level of criticality are unable to interfere with task of higher level of criticality. Period lengths must be non-decreasing, i.e. supporting tasks of higher level of criticality with shorter period than those of lower critical level. This might affect design of tasks by splitting high level critical task into several subtasks.

Determinism is an important feature in real-time systems. With determinism, the next state of the system and associated hardware is predictable. In the SOA this requires that the local cloud need to execute on top of real-time operating system (RTOS), where on other hand other tasks might be scheduled to execute on a generic OS. Analogous, the SOA must hereby transparently manage the mixed-criticality, accordingly to how the real-time tasks are managed in relation to criticality.

The safety services and safe tasks are executed on the Safety RTOS that runs isolated from other non-safe services or software. The non-safe software runs isolated on some regular OS, typical the Linux operating system or Android system.

A common task-model is to divide tasks into groups by their criticality, and that each task has a period, deadline and execution time. For critical services, it hereby requires to include level of criticality, and execution time (or deadline) in the service contract. This defines the type of mixed-criticality properties that a service describes and their constraint on system usage.

2.4 **Binding services and its protocol**

The abstracted service uses a protocol defining the characteristics of the services. These characteristics are then parameters in the process of binding services. The complexity of the protocol depends on how autonomous the binding of services are. In other words services must have a common data model, meaning that services are able to parse, generate and validate data.

Further is it important to keep the data model and the parser simple and compact. This is hereby leading to a requirement for common data formats instead of complex parse/generate tools and validation tools.

Only one solution/component is used for the entire communication. A service also need to parse and generate data over life time, since values often are not static (variables changes over time). Additionally, the service need to validate data i.e. use a schema of two reasons. The first reason is to enable the service consumer to generate messages that a service provider understand, in terms of structure. This will also make sure that the service provider sends correct data types, meaning that the data representation is correct when received and does not inflict any unwanted behaviour. The second reason is that the provider may schema to validate that the messages that are received are correct and do not does not provide any security violation with bogus values.

Additionally, services require a common message framework. Herby the service is able to send messages and interact with any other service. The message will include non-functional properties/elements such as security tokens, maximal execution time etc. As a precaution it is recommended to specify intermediary notes.

To enable services to locate services on basis of what it provides, services must have a service contract. This is needed to verify how the servicer is used and if it can use it. To ensure that the services are able to find service contracts requires that there is a common method to locate service contracts (not just run-time service instances). A simple static method as broadcasting the contracts or more advanced dynamic methods such UDDI. Here is a trade-off between simplicity and sophistication. Safety precaution is to add safety parameters to the service protocol. This is to indicate portioning and isolation.

2.4.1 **Safety services validation**

In the case of the binding of safety services, compounded services, it is required that services are tied together statically and off-line. It is accepted to execute the validation during runtime. When the setup is validated as a safe configuration the compound service is enabled and on-line. The processes hereby require that all valid compound services are assured before released in an online system. Once assured the specific assurance ticket is registered and saves as part of each the service component as part of its protocol. Then when enabled this assurance ticket is verified during the binding procedure. This safe service is then run isolated in its partition.

2.4.2 **Dynamic reconfiguration**

In case of a dynamic reconfiguration of the safe service compound the safety systems must go off-line during the reconfiguration. The new setup is validated and in case of positive result is will go on-line. Any component service that will have any interaction with non-safe service will handle the interaction via a safety proxy service. The safety proxy service is validated together with the services but only act as a robust front to ensure safe interaction with the non-safe system.

2.5 **Local cloud**

The SOA enables a platform a common virtual embedded platform, the fog. The fog will execute one instance on each partition. The virtual layer makes the boundary between partitions invisible and an application running on top will not see the boundaries, regardless if they safe nor non-safe.

2.5.1 **Resource Deployment Management Service**

This component contains resource and communication mechanisms, and may be divided into Communication management, Resource Manager Service, and Middleware Naming Service.

The naming Service enables services to register their service, locate other services, and subscribe to other services. The registry includes one list containing all services in the system, and one list of all subscriptions. When a service subscribes to a specific service, the registry first tries to locate them in the service list. If the registry cannot find it the subscription is saved in the subscription list. If a service subscribes to an available service, it receives a message containing the reference to the service. However, if a service subscribes to an unavailable service, it will be notified about the reference to the service when

the service is available (i.e. when it is added at runtime and registered). The registry also has the ability to remove failed or crashed services from the service list and subscription list.

The Communication Management Service module provides robustness and communication abstraction, and handles incoming and outgoing messages. Furthermore, it encapsulates the fail-safe module and the subscription module. It is responsible for; locating the master service, gather incoming messages and send them to the correct process, synchronizing the system, provide the address to the Naming Service to other services, and remember registrations and subscriptions for fail safe purpose.

The Dependability and QoS component is divided into a Quality of Service (QoS) manager and a Trace and Error manager. It provides the basis in terms of safety, stability, reliability, and security. The QoS manager provides data streaming ability to the services. The Trace and Error manager traces errors and diagnoses the system, either on each node or globally. The information can thereafter be extracted during runtime.

2.5.2 **SOA for mixed criticality and Extended ECU**

The figure below describes the hardware platform with the systems software. The Extended ECU represents the next generation automotive electronics. Instead of having hundreds of ECU as in a modern car, the system is represented by a few ECUs. The new electronic device is not only more powerful but also more flexible to perform various workloads.

The figure visualizes that a system supporting two partitioning, one critical and one non-critical. The system resources are shared but partitioning is supported by both hardware protection and software protection. The virtualization platform may actually be implemented differently depending on the combination of hardware the used features provided by this hard hardware and the virtualization software taking advantage of the provide hardware support. The services are enabled to be transparent (but safety isolated) in the local fog.

The general view of how the embedded cloud (Fog) is shown in Figure 1. The system shows that the multi-core hardware with FPGA, the software platform for virtualization with the operating systems (two partitions one critical and one non-critical) and the platform sensors and I/O. The fog is the dSOA.

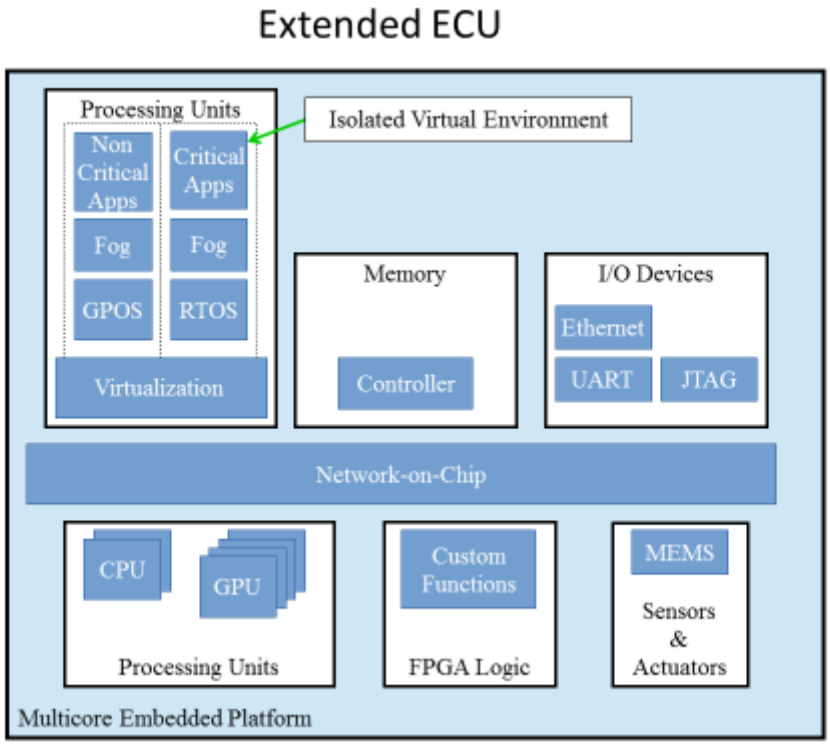

Figure 1: General view of mixed criticality ECU

2.6 Requirement coverage

The following requirements are touched in section 2.

Requirement ID	Short Description
TL-REQ-WP01-001	Service oriented architecture interoperability
TL-REQ-WP01-002	Service access control
TL-REQ-WP01-003	Service discovery
TL-REQ-WP01-004	Addition of new services under operation
TL-REQ-WP01-005	Service composition
TL-REQ-WP01-006	Vertical service deployment
TL-REQ-WP01-007	Data aggregator platform safety
TL-REQ-WP01-008	Data aggregator platform energy usage and communication efficiency
TL-REQ-WP01-009	Data aggregator platform variability and adaptability
TL-REQ-WP01-012	Disjoin of processing units
TL-REQ-WP01-013	Admission control
TL-REQ-WP01-016	Security of reconfiguration
TL-REQ-WP01-019	Communication Services
TL-REQ-WP01-020	Shared Resources/Services
TL-REQ-WP01-021	Deterministic Communication Service
TL-REQ-WP01-043	Security and safety co-consideration
TL-REQ-WP01-059	Mixed Critical Design and verification
TL-REQ-WP01-067	Shared resource identification
TL-REQ-WP01-121	Service validity & Data freshness
TL-REQ-WP01-122	Service latency

3. Conclusions

The report considers the design and principles for runtime assessments. for safe and critical SOA is a potential solution to manage embedded realtime system requirements, dependable requirements, secure requirements and reliable requirements.

The discussed concept and technologies in this report regarding mixed criticality managed by adding isolation and portioning for both temporal as well as spatial, is concluded to be a valid approach. The suggestion on system that provides shared hardware but parted execution is acceptable. The method to handle cross-over communication activities between critical and non-critical execution is managed. Reconfiguration technology to enable critical services is technically doable but it required strong safety design consideration to ensure non-breaking safety assurance regulation.

The design principles on security for dynamic systems are required to ensure safety. The principals are discussed on authorization, permissions, monitoring, enforcement and portability. The assessment is acceptable guideline to ensure safety from the security perspective.

A mixed critical system must be able to provide support for both realtime tasks and non-realtime tasks. The co-existence is an add-on requirement on the mixed-critical system. The recommended guidelines covers the topic with relevance.

4. References

[1] E. Newcomer and G. Lomow, Understanding SOA with Web Services. Hagerstown, Maryland: Pearson Education, Inc., 2004, ISBN: 0-321-18086-0.

[2] A. Eckel, A. Leitner, J. Gustafsson, L.Amorosi, D.Pascucci, R. Pathan, A. Eric, P. Ekdahl, J. Díaz, L. M. Pinho, and W. Axel. (Oct. 2014). Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments. Accessed: 2016-02-11. Publisher: EMC2 Project Consortium. Version 1.1, [Online]. Available: http://www.artemis-emc2.eu/fileadmin/user_upload/Publications/Deliverables/EMC2_D1.1_SotA_and_SOA_architecture_requirements_report_v1_0_Final.pdf.

[3] A. Zisman, G. Spanoudakis, J. Dooley, and I. Siveroni, "Proactive and reactive runtime service discovery: A framework and its evaluation," Software Engineering, IEEE Transactions on, vol. 39, no. 7, pp. 954–974, Jul. 2013, ISSN: 0098-5589. DOI: 10.1109/TSE.2012.84.

[4] J. C. Knight, "Safety critical systems: Challenges and directions," in Proceedings of the 24th International Conference on Software Engineering, ser. ICSE '02, Orlando, Florida: ACM, 2002, pp. 547–550, ISBN: 1-58113-472-X. DOI: 10.1145/581339.581406. [Online]. Available: http://doi.acm.org/10.1145/581339.581406.

[5] EMC2 Consortium. (2014). Ll1 - automotive applications. Accessed: 2016-01-29, [Online]. Available: http://www.artemis-emc2.eu/?id=23.

[6] J. Barhorst, T. Belote, H. Pam Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. (Apr. 2009). A research agenda for mixed-criticalitysystems. Accessed: 2016-01-25, [Online]. Available: http://www.cse.wustl.edu/~cdgill/CPSWEEK09 _ MCAR / RBO - 09 - 130 % 20Joint % 20MCAR % 20White % 20Paper %20PA%20approved.pdf.

[7] J. Bowen, "The ethics of safety-critical systems," Commun. ACM, vol. 43, no. 4, pp. 91–97, Apr. 2000, ISSN: 0001-0782. DOI: 10.1145/332051.332078. [Online]. Available: http://doi.acm.org/10.1145/332051.332078.

[8] G. H. Brundtland, M. Khalid, S. Agnelli, S. A. Al-Athel, B. Chidzero, L. M. Fadika, V. Hauff, I. Lung, M. Shijun, M. M. do Botero, N. Singh, P. Nogueira-Neto, S. Okita, S. S. Ramphal, W. D. Ruckelshaus, M. Sahnoun, E. Salim, B. Shaib, V. Sokolov, J. Stanovnik, and M. Strong, DEVELOPMENT AND INTERNATIONAL ECONOMIC CO-OPERATION, ENVIRONMENT. UN General Assembly, 1987. [Online]. Available: http://www.bne-portal.de/fileadmin/unesco/de/Downloads/Hintergrundmaterial_international/Brundtlandbericht.File.pdf?linklisted=2812.

[9] A. Bahrami, V. A. Clincy, L. Deligiannidis, and G. Jandieri, Eds., Las Vegas, USA: CSREA Press U.S.A, 2013, pp. 67–73, ISBN: 1-60132-243-7.

[10] R. Ramanathan, "White paper: Intel multi-core processors: Making the move to quadcore and beyond," Intel Corporation, United State, Tech. Rep., 2006, Accessed: 2016-03-03. [Online]. Available: https://www.pogolinux.com/learn/files/quad-core-06.pdf.

[11] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, First Edition. Glasgow, Scotland, UK: Strathclyde Academic Media, Jul. 2014, ISBN: 978-0992978709.

[12] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. (Aug. 2006). Reference model for service oriented architecture 1.0. Accessed: 2016-02-16, [Online]. Available: https://www.oasis-open.org/committees/download.php/19679/soa- rmcs.pdf.

[13] Object Management Group (OMG). (Nov. 2008). Common object request broker architecture (corba) for embedded specification. Accessed: 2016-02-17. Version 1.0, [Online]. Available: http://www.omg.org/spec/CORBAe/1.0/.

[14] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, "Building web services with java: Making sense of xml, soap, wsdl, and uddi," in. 201 West 103rd St. Indianapolis, Indiana, 46290 USA: Sams Publishing, Dec. 2001, ISBN: 0-672-32181-5.

[15] World Wide Web Consortium (W3C). (2016). About w3c. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/Consortium/

[16] (2016). W3c mission. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/Consortium/mission

[17] (2015). Web of services. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/standards/webofservices/

[18] (2015). Semantic web. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/standards/semanticweb/

[19] (Feb. 2004). Web services architecture, 1.4 what is a web service? Accessed: 2016-02-18, [Online]. Available: https:/www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis

[20] Microsoft. (Oct. 2014). What is windows communication foundation? Accessed: 2016-02-16, [Online]. Available: https://msdn.microsoft.com/en- us/library/ms731082%28v=vs.110%29.aspx

[21] (Oct. 2014). Web services protocols interoperability guide. Accessed: 2016-02-16, [Online]. Available: https://msdn.microsoft.com/en-us/library/ms734776%28v=vs.110%29.aspx.

[22] (Oct. 2014). Support for json and other data transfer formats. Accessed: 2016-02-16, [Online]. Available: https://msdn.microsoft.com/en-us/library/bb412187%28v=vs.110%29.aspx.

[23] (Oct. 2014). Wcf discovery overview. Accessed: 2016-02-16, [Online]. Available: https://msdn.microsoft.com/en-us/library/dd456791%28v=vs.110%29.aspx.

[24] OASIS. (Jan. 2016). About us. Accessed: 2016-02-17, [Online]. Available: https://www.oasis-open.org/org.

[25] (Jan. 2016). Oasis web services discovery and web services devices profile (ws-dd) tc. Accessed: 2016-02-17, [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-dd.

[26] (Jul. 2009). Devices profile for web services version 1.1. Accessed: 2016-02-17, [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html#_Toc228672077.

[27] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap),"RFC Editor, RFC 7252, Jun. 2014, Accessed: 2016-02-21., pp. 1–112. DOI: http://dx.doi.org/10.17487/RFC7252. [Online]. Available: http://www.rfc-editor.org/info/rfc7252.

[28] Z. Shelby, "Constrained restful environments (core) link format," RFC Editor, RFC 6690, Aug. 2012, Accessed: 2016-02-21., pp. 1–22. DOI: http://dx.doi.org/10.17487/RFC6690. [Online]. Available: http://www.rfc-editor.org/info/rfc6690.

[29] World Wide Web Consortium (W3C). (2015). Xml essentials. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/standards/xml/core.

[30] (2015). Schema. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/standards/xml/schema .

[31] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on, Oct. 2008, pp. 740–747. DOI: 10.1109/LCN.2008.4664275.

[32] C. Groba and S. Clarke, "Web services on embedded systems - a performance study," in Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010, 8th IEEE International Conference on, Mar. 2010, pp. 726–731. DOI: 10.1109/PERCOMW.2010.5470528.

[33] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, "Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture," Industrial Informatics, IEEE Transactions on, vol. 9, no. 1, pp. 43–51, Feb. 2013, ISSN: 1551-3203. DOI: 10.1109/TII.2012.2198655.

[34] World Wide Web Consortium (W3C). (2015). Meta formats. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/standards/webarch/metaformats.

[35] (2014). Rdf 1.1 turtle. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/TR/turtle/.

[36] (May 2000). Simple object access protocol (soap) 1.1. Accessed: 2016-02-15, [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[37] I. Rauf, A. Hanzala Khan, and I. Porres, "Analyzing consistency of behavioral rest web service interfaces," ArXiv e-prints, Oct. 2012. arXiv: 1210.6115 [cs.SE].

[38] World Wide Web Consortium (W3C). (Mar. 2001). Web services description language (wsdl) 1.1. Accessed: 2016-02-16, [Online]. Available: https://www.w3.org/TR/wsdl.

[39] (Aug. 2009). Web application description language. Accessed: 2016-02-19, [Online]. Available: https://www.w3.org/Submission/wadl/.

[40] OASIS. (Jan. 2016). Oasis uddi specification tc. Accessed: 2016-02-15, [Online]. Available: https://www.oasis-open.org/committees/uddi-spec/faq.php.

[41] T. Nixon and A. Regnier. (Jul. 2009). Web services dynamic discovery (ws-discovery) version 1.1. Accessed: 2016-02-16, [Online]. Available: http://docs.oasis-open.org/wsdd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html.

[42] J. Beatty, G. Kakivaya, D. Kemp, T. Kuehnel, and B. Lovering. (Apr. 2005). Web services dynamic discovery (ws- discovery). Accessed: 2016-02-16, [Online]. Available: http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf.

[43] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," RFC Editor, RFC 5246, Aug. 2008, pp. 1–104. DOI: http://dx.doi.org/10.17487/RFC5246. [Online]. Available: http://www.rfc-editor.org/info/rfc5246.

[44] J. Postel, “Transmission control protocol,” RFC Editor, RFC 0793, Sep. 1981, pp. 1–85. [Online]. Available: http://www.rfc-editor.org/info/rfc793.

[45] E. P. Eronen and E. H. Tschofenig, “Pre-shared key ciphersuites for transport layer security (tls),” RFC Editor, RFC 4279, Dec. 2005, pp. 1–15. DOI: http://dx.doi.org/10.17487/RFC4279. [Online]. Available: http://www.rfc-editor.org/info/rfc4279.

[46] E. P. Wouters, E. H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, “Using raw public keys in transport layer security (tls) and datagram transport layer security (dtls),” RFC Editor, RFC 7250, Jun. 2014, pp. 1–18. DOI: http://dx.doi.org/10.17487/RFC7250. [Online]. Available: http://www.rfc-editor.org/info/rfc7250.

[47] I. Grigorik, High Performance Browser Networking: What every web developer should know about networking and web performance. Sebastopol, CA: O’Reilly Media, Sep. 2013, ch. 4: Transport Layer Security (TLS), ISBN: 978-1-4493-4476-4.

[48] J. Kohl and C. Neuman, “The kerberos network authentication service (v5),” RFC Editor, RFC 1510, Sep. 1993, Accessed: 2016-03-01, pp. 1–112. DOI: http://dx.doi.org/10.17487/RFC1510. [Online]. Available: http://www.rfc-editor.org/info/rfc1510.

[49] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, “Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile,” RFC Editor, RFC 5280, May 2008, Accessed: 2016-03-01, pp. 1–151. DOI: http://dx.doi.org/10.17487/RFC5280. [Online]. Available: http://www.rfc-editor.org/info/rfc5280.

[50] J. Postel, “User datagram protocol,” RFC Editor, RFC 0768, Aug. 1980, pp. 1–3. DOI: http://dx.doi.org/10.17487/RFC0768. [Online]. Available: http://www.rfceditor.org/info/rfc768.

[51] E. Rescorla and N. Modadugu, “Datagram transport layer security,” RFC Editor, RFC 4347, Apr. 2006, pp. 1–25. DOI: http://dx.doi.org/10.17487/RFC4347. [Online]. Available: http://www.rfc-editor.org/info/rfc2817.

[52] A. Burns and R. D. I., Mixed criticality systems - a review, Department of Computer Science, University of York, UK, York, Jan. 2016.

[53] R. Obermaisser and B. Leiner, “Temporal and spatial partitioning of a time-triggered operating system based on real-time linux,” in Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, May 2008, pp. 429–435. DOI: 10.1109/ISORC.2008.10.

[54] INTERNATIONAL ELECTROTECHNICAL COMMISSION. (Apr. 2010). Functional safety of electrical/electronic/programmable electronic safety-related systems - part 1: General requirements. Edition 2.0. Accessed: 2016-03-01, [Online]. Available: https://webstore.iec.ch/preview/info_iec61508-1%7Bed2.0%7Db.pdf.

[55] C. Hobbs and P. Lee. (Jul. 2013). Understanding iso 26262 asils. Accessed: 2016-03-01, [Online]. Available: http://electronicdesign.com/embedded/understanding-iso-26262-asils.

[56] J. Anderson, S. Baruah, and B. Brandenburg, “Multicore operating-system support for mixed criticality,” Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, Apr. 2009.

[57] W. Steiner, “Synthesis of static communication schedules for mixed-criticality systems,” in Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW),

2011 14th IEEE International Symposium on, Mar. 2011, pp. 11–18. DOI: 10.1109/ISORCW.2011.12.

[58] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha, "Handling mixed-criticality in soc-based real-time embedded systems," in Proceedings of the Seventh ACM International Conference on Embedded Software, ser. EMSOFT '09, Grenoble, France: ACM, 2009, pp. 235–244, ISBN: 978-1-60558-627-4. DOI: 10.1145/1629335.1629367. [Online]. Available: http://doi.acm.org.focus.lib.kth.se/10.1145/1629335.1629367.

[59] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, Jun. 2010, pp. 1864–1871. DOI: 10.1109/CIT.2010.320.

[60] R. John, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," Tech. Rep., Mar. 1999.

[61] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, "The arrowhead approach for soa application development and documentation," in Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE, Oct. 2014, pp. 2631–2637. DOI: 10.1109/IECON.2014.7048877.

[62] A. Lindell, E. Numanagic, M. Wånggren, and D. Scholle, "Design description shape demonstration platform," Documentation of SHAPE-platform, version 2.1.1, www.alten.se, Feb. 2016.

[63] R. H. Carver and K.-C. Tai, "Modern multithreading: Implementing, testing, and debugging multithreaded java and c++/pthreads/win32 programs," in. Hoboken, NJ, USA: John Wiley & Sons, Inc., Nov. 2005, ch. 5: Message Passing, ISBN: 9780471725046. DOI: 10.1002/0471744174.ch5.

[64] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, "Mpi : The complete reference," in. Cambridge, Mass.: MIT Press, 1996, ch. 2. Point-to-Point Communication, ISBN: 0262691841.

[65] R. H. Carver and K.-C. Tai, "Modern multithreading: Implementing, testing, and debugging multithreaded java and c++/pthreads/win32 programs," in. Hoboken, NJ, USA: John Wiley & Sons, Inc., Nov. 2005, ch. 6. Message Passing in Distributed Programs, ISBN: 9780471725046. DOI: 10.1002/0471744174.ch6.

[66] S. Knox and A. W. Burkard, "Qualitative research interviews," Psychotherapy Research, vol. 19, no. 4-5, pp. 566–575, 2009, PMID: 19579087. DOI: 10.1080/10503300802702105. eprint: http://dx.doi.org/10.1080/10503300802702105. [Online]. Available: http://dx.doi.org/10.1080/10503300802702105.

[67] G. Partington, "Qualitative research interviews: Identifying problems in technique," Issues In Educational Research, vol. 11, no. 2, pp. 32–44, 2001, Accessed: 2016-02-08. [Online]. Available: http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=5367&context=ecuworks.

5. Abbreviations

Table 1: Abbreviations

Abbreviation	Meaning
EMC^2	Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments
µKernel	Micro-Kernel
SOA	Service Oriented Architecture
WSDL	Web Services Description Language
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
ECU	Engine Control Unit or Electronic Control Unit