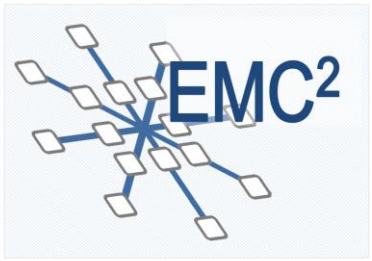

Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D1.8 – Dependability services of the EMC² architecture**	
Work package	WP1	SOA Embedded Systems Architecture
Task / Use Case	T1.6	Safety and Fault-tolerance Concept
Subtasks involved	T1.1 Requirements and Specification	
Lead contractor	Infineon Technologies AG Dr. Werner Weber, mailto:werner.weber@infineon.com	
Deliverable responsible	Alten Detlef Scholle, detlef.scholle@alten.com	
Version number	v1.6	
Date	27/04/2016	
Status	Final	
Dissemination level	PU / CO (Section 7)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
02F	VIF	Mario Driussi Helmut Martin	All Chapters + T1.6 Internal Report
11A	Alenia	Marco Terrone	Chapter 7.2/7.6 + T1.6 Internal Report
02C	IFAT	Stefan Berger Alexander Lipautz	Chapter 7.1 + T1.6 Internal Report
16A	Chalmers	Ioannis Sourdis	Chapter 7.3 + T1.6 Internal Report
15O	SevenS	José Luis Gutiérrez	Chapter 7.5 + T1.6 Internal Report
16M	Alten	Detlef Scholle	Chapter 7.6

Document History

Version	Date	Author name	Reason
V0.01	06/03/2016	Mario Driussi	Initial draft
V0.02	11/03/2016	Mario Driussi	Updated chapter 1-9
V0.03	20/03/2016	Mario Driussi	Updated chapter 1-9
V1.0	21/03/2016	Mario Driussi	Updated chapter 1-9
V1.1	22/03/2016	Marco Terrone	Chapter 8
V1.2	23/03/2016	Mario Driussi	Updated all chapters
V1.3	24/03/2016	Helmut Martin	Update of document
V1.4	01/04/2016	Atul Yadav	Review of document
V1.5	06/04/2016	Mario Driussi	Revision of document after review
V1.6	26/04/2016	Jan Deventer	Release
	27/04/2016	Alfred Hoess	Final editing and formatting, deliverable submission

Publishable Executive Summary

The scope of this document is to provide information about dependability and Service Oriented Architectures (SOA) for multi-core devices.
The goals are:

- To integrate embedded multi-core devices in a safe and secure manner, because existing Service Oriented Architectures (SOAs) are not sufficient in their characteristics in context of dependability requirements demanded by critical applications.
- To investigate existing SOA concepts and elaborate additional requirements for a dependable Service oriented Architecture (dSOA).

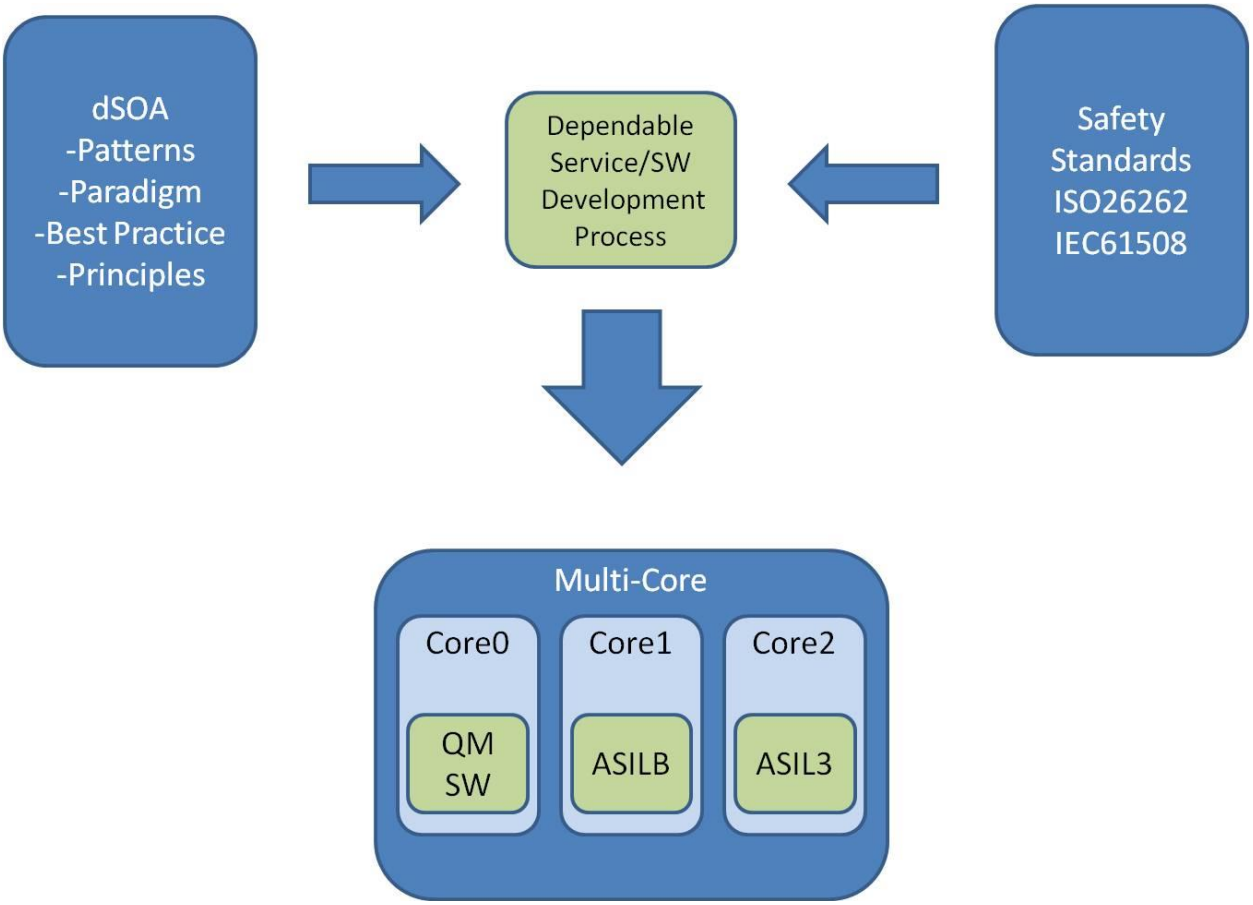

Figure 1: SOA and safety standards have to be merged in order to run dependable services on a multi-core device

Table of contents

List of figures

List of tables

1. Objective and scope of the document

In a highly interconnected world of (embedded) devices, also known as the Internet of Things (IoT), more and more safety critical devices and applications are getting interconnected. The goals are:

- To integrate embedded multi-core devices in a safe and secure manner, existing Service Oriented Architectures (SOAs) are not sufficient in their characteristics in context of dependability requirements demanded by critical applications.
- To investigate existing SOA concepts and to work out additional requirements for a dependable Service oriented Architecture (dSOA).

Security topics investigated in Sub-Task 1.5 and dynamic runtime behavior investigated in Sub-Task 1.4 are out of scope for the work here.

1.1 Structure of the deliverable report

We start with an introduction in chapter 2 and an overview about general SOA concepts and related architectures form the automotive domain (AUTOSAR and SOME-IP), followed by definition of dependability in general in chapter 3. Chapter 4 provides a list of general required services and a short description of each service for a dependable Service oriented Architecture (dSOA). Chapter 5 explains the main barrier to build a dSOA followed by a conclusion in chapter 6. Chapter 7 provides information about partner contributions and chapter 8 gives an overview about the related Living Labs in the EMC2 project. Chapter 9 provides links with references. Chapter 10 “Appendix A” provides additional information about the related working document of Sub-Task 1.6 in WP1 in the document “EMC2_T1.6_InternalReport_v1.0”, containing more detailed information about the specific partner contributions concerning safety and fault-tolerance.

1.2 Requirements

Task T1.1 has collected requirements from the other Work Packages and Living Labs. From this list, we have extracted the relevant architectural dependability and added them to the corresponding services in chapter 2. The complete list of the technical requirements is available at EMDesk [19].

2. Introduction

Today about 90% of the daily computation power is consumed on embedded devices (such as cell phones, smart homes, pacemakers, etc.). These embedded devices interact with the physical world. The software complexity grows up in this highly interconnected world. Such complex software is error prone and bugs are unavoidable. To reduce complexity and provide an infrastructure new software architectures are needed. Service oriented Architecture (SOA) is one beneficial approach to handle the complexity and to avoid/reduce bugs. SOAs are widely spread across the World Wide Web (WWW) applications. Web Services (WS) for example are standardized by the World Wide Web Consortium (W3C) [1] and prosper in several business application systems like Enterprise Business Services (EBS).

Furthermore, SOAs have several properties, which make them very powerful and successful. These properties are based on design patterns and best practice methods, which are well known from different software development and development life cycle processes. That makes them very efficient with respect to reusability and time to market aspects.

Classic SOA consists of a Service Registry/Repository (Service Broker), Service Providers and Service Consumers. A service contract belongs to each service (see Figure 2).

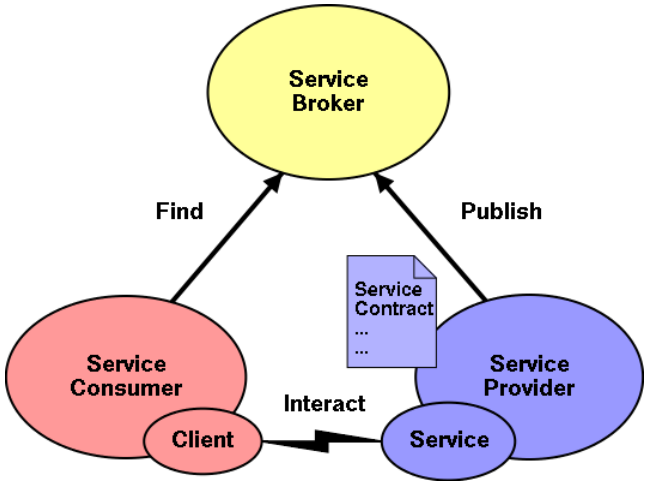

Figure 2: Simple Service oriented architecture from W3C[2]

General SOAs originate not in the context of safety critical applications running on industrial embedded multi-core systems. The roots of SOA and the main requirements are coming from business application development. Commonly we see that services are unassociated, loosely coupled units of functionality that are self-contained.

A service description contains all required information to understand the functionality and serve as implementation specification. The service contract expresses the service capabilities. Based on the contract every other service (developer) knows how to use it, and how the interfaces look like (formal interface description). Furthermore, services should be loosely coupled and stateless. The objective for loose coupling is to reduce bindings between services, so that one service can be updated or changed in a manner that such a change or update does not break the existing relationship between a compound of services which are building a system. Statelessness supports this in order to design scalable services by separating them from their state data whenever possible.

Services are also discoverable and composable. Discoverability supports reusability because services can be found and used by its customers via a repository. In a System or a System of Systems several services are interlinked and build a new system based on their compensability.

[1] https://www.w3.org/Consortium/
[2] https://www.w3.org/2003/Talks/0521-hh-wsa/slide5-0.html

Nevertheless, for embedded multi-core systems with real-time constrains and safety requirements additional properties to design a dependable Service Oriented Architecture (dSOA) for an embedded multi-core system are needed. SOA provides two approaches for the development of a service. The code first approach and the contract first approach. For a dSOA the contract first approach should be applied. Since it is very hard to find the right method(s) for assessment of an existing services or systems.

A System based on a SOA, which has real-time, safety and security constraints is the AUTomotive Open System ARchitecture (AUTOSAR).

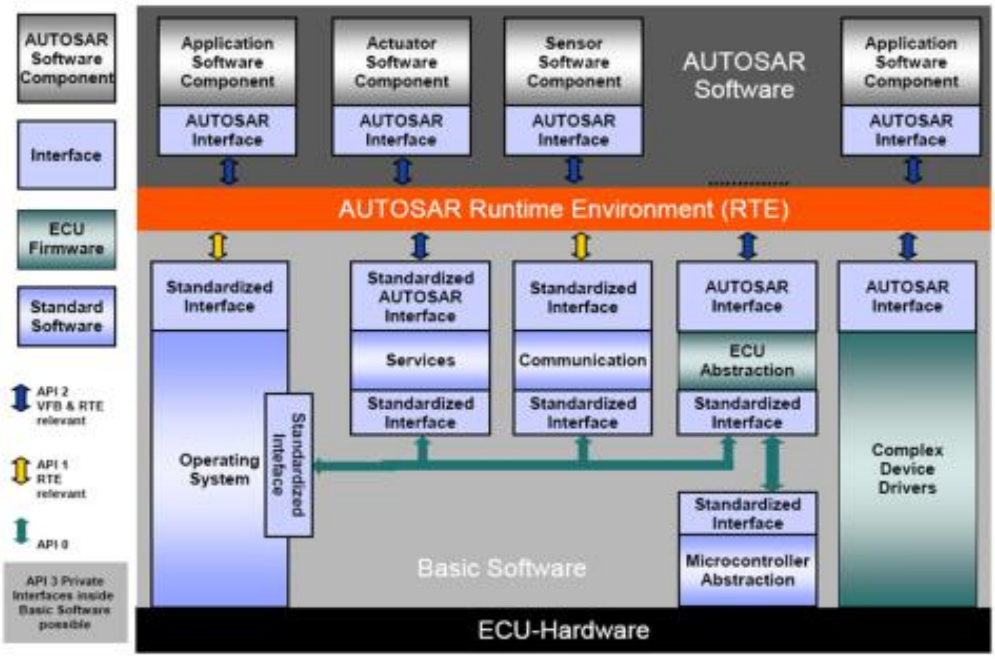

Figure 3: AUTOSAR Software Architecture – Components and Interfaces[3]

AUTOSAR is an approach to use the benefits of service orientation, however it has some important restrictions towards SOA as used and applied for Web Service (WS). One important restriction is that dynamic behavior at runtime is not supported whereas dynamic behavior at runtime is one of the key features from a SOA. Nonetheless, a static design at runtime has important advantages with respect to safety and real-time constraints. For example, a static system can be analyzed and configured offline to guarantee the required runtime behavior when the system is online. That is for safety critical real-time applications an important case.

AUTOSAR offers some mechanisms for dynamic replacement of functions. RTE mode switches, starting and stopping alarms and tasks (to influence performance), starting and stopping OS applications but this needs a static configuration before build time.

Nevertheless, AUTOSAR supports since version 4.x with the SOMEIP Stack (Scalable service-Oriented MiddlewarE over IP) a service oriented concept. SOMEIP is an automotive middleware solution designed to run on different operating systems and devices. SOMEIP supports Remote Procedure Calls (RPC) Service Discovery (SD), publish and subscribe (Pub/Sub) which can be seen as typical service oriented features. A system supporting SOMEIP and AUTOSAR is still static at runtime. This is based on the used "Open Systems and their Interfaces for the Electronics in Motor Vehicles" (OSEK) conform operating systems, which are designed to run on devices with very application specific resources. Such kind of automotive ECU has neither explicit RAM nor several caches. To implement a SOA as known from WS

[3] autosar.org

on an embedded multi-core device as used in the automotive domain is due to its restricted HW resources not possible.

For this, the future AUTOSAR group works on the Adaptive AUTOSAR Platform which supports with Portable Operating System Interface (POSIX) an interface that allows running much more powerful operating systems. With this approach general purpose CPUs are coming in the focus. With the computation power of such devices, a SOA concept is the logical consequence. With respect to safety, real-time and fault-tolerance the development life cycle needs to be adapted. AUTOSAR will develop and support both approaches. The classical AUTOSAR Stack which is a static system and runs on an application specific hardware and the adaptive AUTOSAR platform, which provides the demanded computation power for future car applications e.g. autonomous driving functions that support dynamic behavior and runs on a SOA. To fulfill safety, real-time and fault-tolerance requirements a dSOA is needed.

3. Dependability

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 [18]. 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 we define the following attributes for a 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 (see D1.6)
- 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 (see D1.6)

Threats are:

- Faults – incorrect step, process or data information
- Errors – discrepancy between a computed and specified value
- Failures – event that occurs when delivered service deviates from correct service

Impairments are:

- Attacks (see D1.6)

Means are:

- Fault Prevention
- Fault Tolerance
- Fault Removal
- Fault Forecasting.

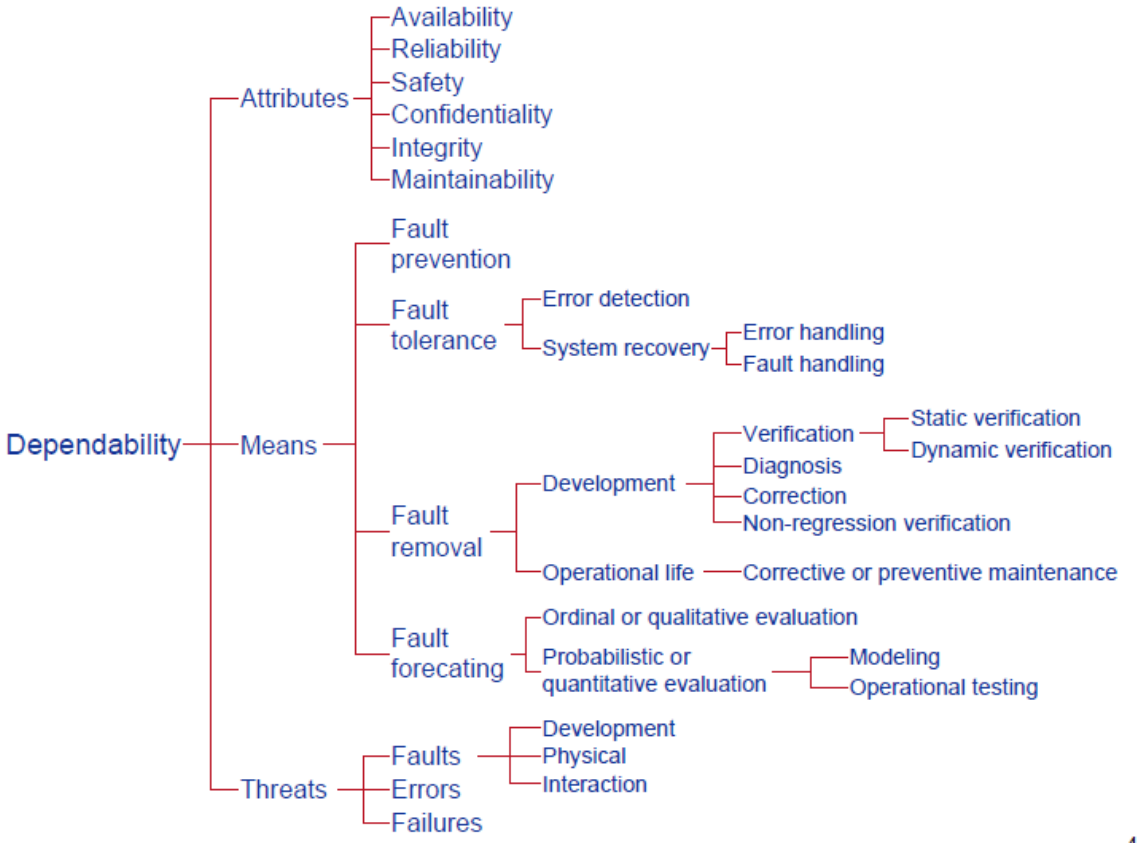

Figure 4: Overview dependability tree from "Dependability and Resilience by Jean-Charles Laprie"

In the context of T1.6 the main focus is set on fault-tolerance. Nevertheless, Fault Prevention, Removal and Forecasting are important means in the context of a dSOA. It is arguable that a dSOA has additional requirements to the executed services itself as well as to the environment (middleware, operating system, target platform, framework) where they are running on, in contrast to a SOA used for web services.

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 threats. To expose the system threats a Design Reference Mission (DRM) and a HAZOP have to work out on the systems use case. That could be seen as a general requirement. To design a dSOA the whole Development Life Cycle (DLC) has to align to an adequate standard. The ISO 26262 [5] and the IEC 61508 [3], which are generally used in automotive and industrial domains, are not sufficient, because they require that all components already are know by the development phase. For a dSOA, which builds on existing services (components) it is a contradiction. The whole architecture must be developed under adequate standards that, fulfills the dependability requirements. That encompasses the application services, the middleware and the operating system as well as the hardware platform.

Required Services for a dSOA without security services (see D1.6) are:

- Dependable simple service (application) and dService contract
- Configuration launcher service
- State manager service
- Execution manager service
- Communication manager service
- Mode manager service
- Memory protection service
- Error detection service

- Fault detection service
- Monitoring service
- Resource manager service
- Requirements validation service
- Diagnostic manager service
- Reconfiguration manager service
- Software manager service

Furthermore, the operating system has to provide adequate interfaces like POSIX and functionality. Service descriptions have to be developed under dependability aspects and service contracts need an extension with dependability attributes.

4. Dependable simple service and dService contract

A service developed under considerations of dependability has to provide additional information to the middleware, the operating system or the environment to get the required execution time.
Presumed that the service development lifecycle is aligned to an adequate standard and the service has safety requirements, for example an ASIL (Automotive Safety Integrity Level), real-time constraints (e.g. 10ms cyclic task), or the communication has real-time constraints, the infrastructure has to be aware to deal with this. For a dependable simple service, this affects primarily the service contract, which provides information about interfaces and the functionality.

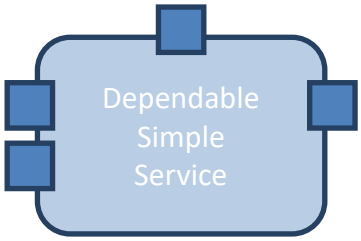

Figure 5: Dependable Simple Service

The service is like a black box, which communicates with its environment via interfaces. The information "how to use" the specific service and a description about the functionality have to be provided with a referenced dService contract. That means additional attributes have to be included in the contract based on a standardized format or an additional contract with the dependability attributes has to be added to the dependable service or the service contract.

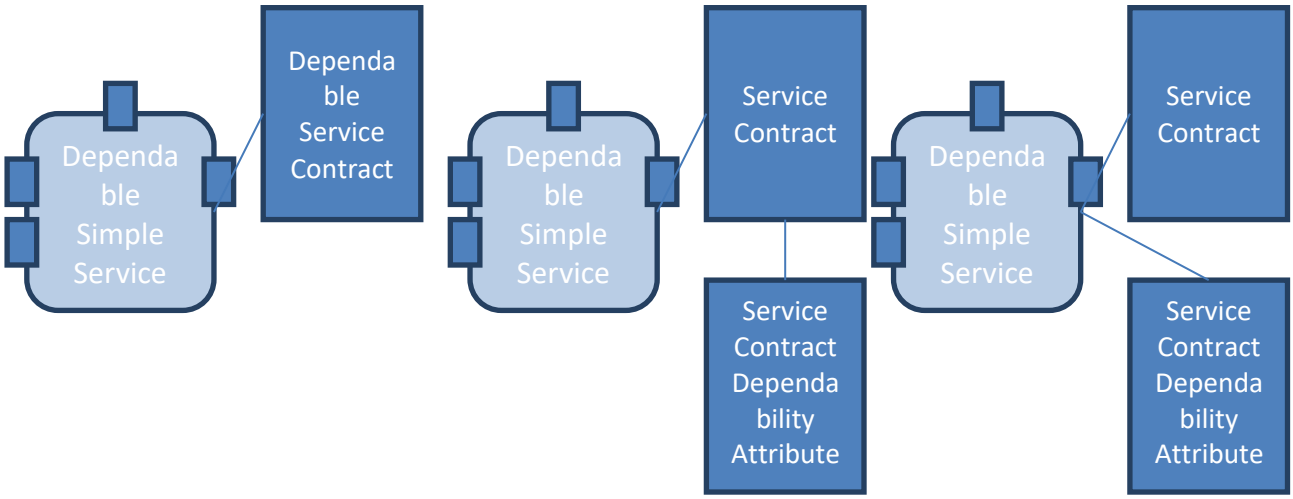

Figure 6: Dependable Simple Service with dService Contract

Dependable service contract:

```
<element name =”Safety”>
      <attribute-Safety>ASIL D
      <attribute-Safety>Process Isolation
      <attribute-FT>Fault-tolerance
            <attribute-FT>RtB
            <attribute-FT>TMR
            <attribute-FT>2oo3D
            …
</element>
<element name = “Execution”>
      <attribute>10ms
      <attribute>cyclic
      …
</element>
<element>Communication
      <attribute>TCP/IP
      <attribute>secure com
      <attribute>end to end protection
      …
</element>
```

Another important difference between a classic SOA concept and a dSOA for embedded multi-core devices is that classical SOAs support a dynamic runtime behavior. That means services can be started, stopped, installed, uninstalled and updated at runtime. OSGI Alliance (former Open Standard Gateway Initiative) standardizes a hardware independent service platform and provides an open standard. The OSGI specification describes a modular system and service platform that implements a dynamic component model. Application or components (services) coming in the form of bundles for deployment, can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot. There are security features implemented but safety, fault tolerance or real-time constraints are not specified.

In the domain of dependability and embedded devices, this is an upcoming approach but it is hard to guarantee runtime behavior. For this appropriate Probabilistic Risk Assessment (PRA) methods have to be used in the dSOA concept phase and for runtime assessment of the system.

The following requirements are covered by this section:
TL-REQ-WP01-064, TL-REQ-WP01-065, TL-REQ-WP01-074

4.1 Configuration launcher service

The start up sequence of a multi-core system is a very critical process with respect to safety and security aspects. For a secure boot process see D1.6 concerning security services. Concerning safety, this means that the right configuration must be loaded to guarantee expected runtime behavior. Available hardware must be checked and initialized. In a dependable system set-up, different configurations with respect to safety must be supported. For freedom of interference that could be memory partitioning and process isolation. Memory partitioning provides protection by means of restricting access to memory (see section 4.6 “Memory protection service”).

A hypervisor setup is another approach that manages the hardware, separate cores, and hardware resources, and hosts different operating systems. A typical hypervisor set-up could run a real-time operating system like AUTOSAR OS on one core and a standard Linux system on the other core. The hypervisor manages the shared hardware for the so-called guest systems.

The following requirements are covered by this section:
TL-REQ-WP01-014, TL-REQ-WP01-039, TL-REQ-WP01-054, TL-REQ-WP01-058, TL-REQ-WP01-133

4.2 State manager service

Web Services should be developed stateless whenever it is possible. This approach is a contradiction to safety related services. Possible service states are important in a safety analysis. They are also important for fault-tolerance. If a stateless service is reused in a new system it could be harmful for the whole safety concept.

A state manager should know the actual state of a service in a system as well as the last state. If a service uses another local service, the state manager should know the states of both services. If a service uses external services, devices, sensors or actuators the state manger should be aware about their states. This could be a state manager to state manager communication, or if necessary one or more global state managers that are responsible to be aware about the whole system state. Based on the state of observed services the state manager provides this information to the mode manger.

Other infrastructure services should also provide their states to the state manager service. The execution manager service should also provide information about services under his administration (e.g. service running, ready, waiting or stopped). The Memory protection manager should provide states about failed memory initialization, read or write failures, memory test results at start up, as bad marked memory areas. The resource manager should provide the resource usage. If there is a state manger depends on the specific application, the architecture and the safety goals.

The following requirements are covered by this section:
TL-REQ-WP01-066, TL-REQ-WP01-067

4.3 Execution manager service

The execution manager is responsible for providing execution time to services. Typical the execution manager is part of the operating system. In a dSOA special execution manager(s) should be provided to support the execution of different kind of services. For example, an ".exe" file is executed by the windows operating system, for a java-based implementation a java virtual machine is required to execute the file. The service contract should include detailed information about execution time and required resources. For a safety critical system, adequate runtime analysis methods should be used (for example "Worst case execution time analysis").

The execution manager should provide an intra process communication between tasks, threats and processes running on different cores, like AUTOSAR OS applications.

Future dSOA design patterns should provide online analysis and reconfiguration in a safe and secure manner (e.g. use of PRA and other methods for online certification and runtime assessment).

The following requirements are covered by this section:
TL-REQ-WP01-018, TL-REQ-WP01-069, TL-REQ-WP01-072, TL-REQ-WP01-104

4.4 Communication manager service

Safe and secure communication is one of the main challenges in a dSOA. For security related communication, see D1.7 security services. Concerning dependability in general, this means that the communication paths have to be available. If necessary, communication paths have to be designed with redundancies. The communication manager is responsible for keeping communication paths alive and serving high priority massages preferred.

That means if an Ethernet connection is established between two devices a redundant design requires two switches. To serve high priority messages first the communication manager has to provide functionality to fulfill this requirement. That could be implemented with three different queues for example. One queue for high priority messages one for lower priority and one for messages without safety or timing constraints. Most communication stacks provide some kind of communication management. Existing

communication protocols like CAN and FLEXRAY will be stretched to their limits, because they need a static configuration. The communication manager is also responsible for the reconfiguration of communication paths.

The following requirements are covered by this section:
TL-REQ-WP01-056, TL-REQ-WP01-057

4.5 Mode management service

Depending on the criticality of the application a dSOA must support different modes. That means a normal mode if the system is running as specified, or in case of a failure, error or fault the system must change the mode to fail-safe, fail-secure, fail-operational, fail-passive to ensure graceful degradation.

4.6 Memory protection service

Memory Protection is important for safety, as well as security. Basically, it is a method for preventing processes or users from accessing memory which is not allocated to them. The security related aspect of this service is rather obvious: Malicious intentions or "illegal" user input is blocked with this method. Especially for systems, which are interconnected and have access to the Internet, this becomes an important issue.

However, memory protection also plays an important role in terms of functional safety. It is for example a helpful tool in the development process, because "illegal" behavior of erroneous services can be easily identified and taken care of. Even more important, it serves as error detection and –containment mechanism, preventing an error in a single service to propagate and infect the whole system.

The following requirements are covered by this section:
TL-REQ-WP01-039, TL-REQ-WP01-040

4.7 Error detection service

A typical error detection service is the Watchdog Timer. The Watchdog Timer is capable of detecting a fault, as well as some control flow errors. However, most of the errors remain unnoticed. There are several approaches in the design of an Error Detection Service, but the most simple and approved one, is the so-called Triple Modular Redundancy (TMR). With this approach, an Error Detection mirrors three individual and independent services, which provide the same functionality. With respect to automotive applications, this could be, for example, services for acceleration measurement. A comparing logic inside the Error Detection Service compares the information received from these services and can identify an error in one of the services by means of a simple voting mechanism.

In advance, another service can be triggered for restarting the service in question, or other actions.

The following requirements are covered by this section:
TL-REQ-WP01-123

4.8 Fault detection service

Which fault detection or recovery service are implemented, depends on safety concepts and design decisions. According to the standard ISO 26262, a *fault* is an "abnormal condition that can cause an item to fail". A *failure*, in turn, is the incapability of an item to provide its intended functionality. In other words, it means that the item is broken down or provides erroneous data. An *error* is the deviation of a computed, observed or measured value from the theoretical correct value.

Electric systems suffer from numerous different possible faults, which increase with the complexity of the system, including, not only, internal faults like design-faults or hardware wear-outs, but also external faults (e.g. example misuse by the user or cosmic radiation).

Due to billions of transistors present in advanced electric systems and the various circumstances that lead to faults, faultless systems remain a utopian dream. Thus, the fault tolerance mechanisms are not actually concerned with the prevention of faults, but assure that a fault does not lead directly to the violation of the safety goal(s), which maintain the system in a safe state. This can be referred to by the term "fail operational" and means that a system is working correctly and reliable, even in the presence of a certain number of faults. This is, more or less, the key concept of any fault tolerant system.

Failure performance of redundant systems

If a failure in a redundant system occurs, the failure performance can be differentiated in three modes:

- Fail-Safe. Become safe when they cannot operate.
  In case of a failure, the system is no longer available and it will switch to a well-defined system safe state at its outputs. The failure of a component has to be handled by additional counter measures that lead to the controllable result (e.g. switching from an automatic mode to a manual mode to override any faulty automatic command).
  Examples: Many medical systems fall into this category.

- Fail-Passive. A system failure "does no harm".
  The entire system has to consist of two fail-safe sub-systems and there are fault-detection and fault-inhibition measures available. Both sub-systems have to compare their output signals. In case of any difference, the resulting output signal has to be zero, which means that the entire system behavior is passive.
  Example: aircraft autopilots that stop controlling the plane, but won't steer aircraft in the wrong direction.

- Fail-Operational. Continue to operate when they fail.
  The entire system is still operating in any case of failure and the system will not change to a fault-state. The entire system has to consist of three sub-systems, which have to provide a fault diagnostic and fault inhibition service. By using comparison of the three sub-systems a faulty behavior in any of the sub-systems should be detectable. Such kind of system-architecture is called fault-tolerant. Fault-tolerant systems avoid service failure when faults are introduced to the system.
  Examples: elevators, the gas thermostats in most home furnaces, and passively safe nuclear reactors.

There are several strategies and concepts in order to make a system "fail operational":

- Error Detection. An error is the deviation of a computed, observed or measured value or condition and the expected, theoretically correct value. An error occurs as consequence of unexpected operating conditions or a fault. The detection of errors is usually done by comparing the results of redundant elements.

- Error Containment. Error containment is supposed to prevent the propagation of an error across specified regions. In the actual implementation, the error containment is conducted by means of ECRs (Error Containment Regions).

- Error Masking. This denotes the dynamic correction of generated errors at runtime and requires multiple redundant systems and voting systems in order to do that. An example of an error masking mechanism, which is also provided by the GENESYS architecture, is TRM (Triple Modular Redundancy). With this method, three independent components execute the same functionality. The output is then processed by a majority-voting system in order to produce a single output. If any of the

three systems is erroneous, the other two are able to correct and mask the error. From the outside, it is not even visible that a fault occurred[9].

- Diagnosis. This is used for identification of a faulty element, which is responsible for an error. Like the error masking, this can be conducted by redundant elements whose output is compared by a voting system.

- Recovery. Recovery is the return of a system to a state, where it able to operate according to its specification. In the simplest case this means just restarting a faulty component. Prerequisite therefore is the detection and location of corrupt elements. There exist backward and forward recovery techniques.

In the following three major fault tolerant designs for recovery of software-intensive services are introduced:

- Recovery block (RcB) is a backward recovery technique that uses the acceptance test (AT) to check the outputs of a module; the next alternative is invoked once the former alternative fails the AT. The executive in RcB is responsible for check point establishment, checkpoint restoration, invocation of the alternatives, and successful return of RcB. RcB is vulnerable to failure under conditions such as when (1) checkpoint establishment fails, (2) checkpoint restoration or invocation of the next alternative fails, or (3) the AT itself fails or no alternative passes the AT.

- N-Version Programming (NVP) is a forward recovery technique that uses the decision maker (DM) to vote for the outputs of all the involved alternative modules. The executive is responsible for input distribution and successful return of the NVP block. NVP is vulnerable to failure under conditions such as when (1) input distribution to the alternatives fails, (2) less than $\lfloor n/2 + 1 \rfloor$ consistent and correct results successfully reach the DM, or (3) the DM itself fails.

- Retry Block (RtB) is a backward recovery technique that also uses AT like RcB to check the outputs of a module. Contrary to RcB, RtB retries the same module rather than using another module once the AT evaluation fails. The original design of RtB requires a data re-expression algorithm (DRA) to change the form of inputs before reusing the same module, but since in the Webapplications or SOA systems failures may be caused by temporary service busy or network congestion and it is not always possible to tailor the DRA for the application, some designs eliminate DRA and use the same inputs on retry, and this simplified retry mechanism is widely supported in modern SOA execution environments. This paper considers the latter designs. RtB is vulnerable to failure under conditions such as when (1) checkpoint establishment fails, (2) checkpoint restoration fails, or (3) the AT itself fails or the retry limit is exceeded before the AT passes.

A Fault Detection Service (FDS) is a service that is capable of detecting faults, and eventually, depending on its implementation, also “control flow errors”. Control flow errors are errors, which lead to a wrong execution sequence of the instructions of a service.

Technically, the FDS can be implemented as simple timer circuit with a specified threshold time. If this limit is reached, it changes its state, which triggers further actions, like restarting a component or activating another safety service.

The advantages of the FDS are the simple design, which reduces the additional complexity of the overall system, as well as the costs. Concerning the functional principle, there are different designs with increasing complexity, which can provide, for example, a certain time window for the response. The well-known implementation of this service is the Watchdog Timer (WDT).

The following requirements are covered by this section:
TL-REQ-WP01-042, TL-REQ-WP01-057, TL-REQ-WP01-075, TL-REQ-WP01-098, TL-REQ-WP01-099

4.9 Monitoring service

The monitoring service is responsible for supervising all running services and for the notification of deviant behavior. The monitoring service verifies our claims with respect to dependability.

The following requirements are covered by this section:
TL-REQ-WP01-010, TL-REQ-WP01-023, TL-REQ-WP01-117, TL-REQ-WP01-134

4.10 Resource manager service

The resource manager service is responsible for resource management on the local system. If a new service is launched for example the resource manager should check availability of required resources based on the service contract and report resource usage in the lifecycle. The Resource manager should provide information about resource usage to the state manager to share the resource usage information with other devices and systems in a safety critical environment.

The following requirements are covered by this section:
TL-REQ-WP01-010, TL-REQ-WP01-015, TL-REQ-WP01-066, TL-REQ-WP01-067,
TL-REQ-WP01-117, TL-REQ-WP01-134

4.11 Requirements Validation:

This service should be responsible for checking the compliance of safety margins, when orchestrating services to higher level services. For example, if a service is required to be in compliance with ASIL D, it cannot be based on another service, which can only provide ASIL B.

According to its specification the requirements validation service could either prevent this orchestration, or even look for possible solutions, e.g. looking for a service with the same functionality, but ASIL D, or combining two of the ASIL B services.

The following requirements are covered by this section:
TL-REQ-WP01-039, TL-REQ-WP01-041, TL-REQ-WP01-045

4.12 Diagnostic manager service

Online and offline diagnostic services for software and hardware are required for maintenance. Diagnostic tests can provide significant improvements in the achievement of dependability of a system. Diagnostic tests should be simple and resource sparing, otherwise system complexity and overhead grows up.

The following requirements are covered by this section:
TL-REQ-WP01-109, TL-REQ-WP01-117, TL-REQ-WP01-134

4.13 Reconfiguration manager service

If a dynamic runtime behavior is supported, online runtime assessment is required (see D1.9). Reconfiguration could be also necessary in term of fault-tolerance with respect to backup recovery for example. A planed reconfiguration should be tested offline to check the consequences on the system behavior with respect to safety and security and relationships to external systems.

The following requirements are covered by this section:
TL-REQ-WP01-017, TL-REQ-WP01-041, TL-REQ-WP01-057, TL-REQ-WP01-080

4.14 Software manager service

If software updates are allowed and services can be started, stopped, installed, uninstalled, updated, and upgraded at runtime a service load manager is required. If new software is loaded or available the resource manager has to check if enough resources are available, the safe requirement manager has to check the safety requirements provided by the service contract and the reconfiguration manager has to reconfigure the system and update communication path configuration, execution paths, and memory alignment.

If a new or an updated service is available, the software manager has to provide information about the contract which includes information about dependability (Unique ID safety contracts, resources, etc.) to the related safety managers. If the service is accepted, the service can be downloaded.

The following requirements are covered by this section:
TL-REQ-WP01-041, TL-REQ-WP01-076, TL-REQ-WP01-077

4.15 Modular Design

A modular design for a dSOA is, with respect of reusability and general SOA design principles, one of the key features in terms of efficiency and time to market.

The following requirements are covered by this section:
TL-REQ-WP01-035

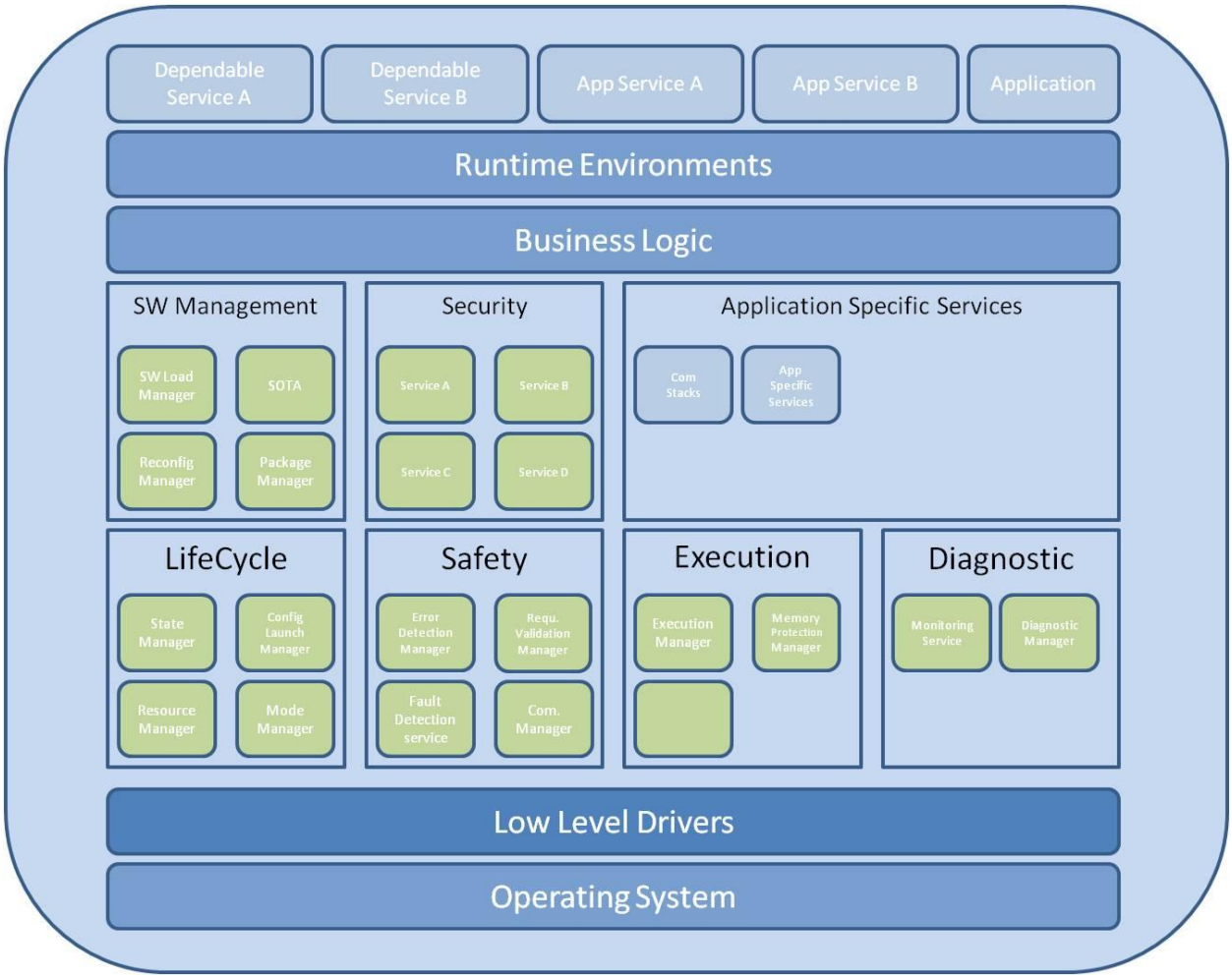

Figure 7: General dSOA overview, green blocks supports dependability

5. Development lifecycle process

The services required for a dependable SOA are implemented and available in several safety critical embedded systems with different characteristics and range of functionality. These implementations however are domain and application dependent. One of the consequences is that they cannot be reused as suggested by SOAs best practice methods and patterns. Furthermore, they are implemented based on standards and target devices, which are also domain dependent.

The gap to run a dSOA on an embedded multi-core device(s) which is applicable for different domains and criticalities is, that there is no domain independent dependable development / system lifecycle process available which fulfills requirements like dynamic runtime assessment, or assessment of composeable systems with real-time, safety and security constrains.

Such a generic process for dependable service oriented architectures is required. This process must support all phases of a product life cycle from the commissioning phase to the decommissioning phase. This process should provide methods for online certification and runtime assurance to achieve the required system dependability with respect to safety and security in a dynamic dSOA implementation. A (general) operating system standard, which supports dependable and dynamic runtime behavior is also required.

The following requirements are covered by this section:
TL-REQ-WP01-044, TL-REQ-WP01-045, TL-REQ-WP01-046, TL-REQ-WP01-047, TL-REQ-WP01-068

6. Conclusions

Not all of the services listed above are stringently necessary. In the design phase of an application and after a safety and security analysis, when the system safety and security requirements and the functional safety and security requirements are on the table a reasonable architecture can be defined. Based on the outcome and verification the required services can be integrated. Note that not every system with dependability requirements can be implemented as service oriented architecture.

Real-time, safety and security or dependability aspects are not in the scope of actual SOA concepts and implementations as well as safety or security standards does not support general SOA patterns and best practice methods.

If a SOA should be implemented for a system with dependability requirements, the whole system / product lifecycle must be aligned to an adequate dependability standard. Even the infrastructure and the environment need appropriate assurance and have to be developed aligned to an appropriate standard.

7. Partner contributions (consortium confidential part of the report)

8. References

[1] Web Service Specifications https://www.w3.org/2002/ws/Reference no. 2

[2] SOME/IP http://some-ip.com/

[3] IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES) http://www.iec.ch/functionalsafety/

[4] Basic concepts and taxonomy of dependable and secure computing http://www.nasa.gov/pdf/636745main_day_3-algirdas_avizienis.pdf

[5] ISO 26262 road vehicles – functional safety – part 1: Vocabulary, 2011.

[6] Obermaisser, R., Kopetz, H., *GENESYS – A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*, Vienna University of Technology, 2009.

[7] Nelson, V., "Fault-tolerant computing: fundamental concepts," IEEE Computer Society, 1990.

[8] AUTOSAR, "Glossary, AUTOSAR Release 4.2.1," 2014.

[9] Wikipedia Contributors, "Triple modular redundancy," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Triple_modular_redundancy, [Visited July 2015].

[10] Yamada, S., Nakamoto, Y., et al., "Generic memory protection mechanism for embedded system and its application to embedded component systems," Computer and Information Technology Workshops, 2008, IEEE 8th International Conference on. IEEE, 2008.

[11] El-Attar, A.M., Fahmy, G., "An Improved Watchdog Timer to Enhance Imaging System Reliability In The Presence Of Soft Errors," Signal Processing and Information Technology, 2007 IEEE International Symposium on. IEEE, 2007.

[12] Lund, E., "EMC2 Service architecture," 2015

[13] Peng, Kuan-Li, and Chin-Yu Huang. "Reliability Evaluation of Service-Oriented Architecture Systems Considering Fault-Tolerance Designs." Journal of Applied Mathematics 2014 (2014).

[14] IEC 62439, Industrial communication networks – high availability automation networks.

[15] "White rabbit: Sub-nanosecond timing distribution over ethernet" Moreira; P. Serrano, J. ; Wlostowski, T. ; Loschmidt, P. ; Gaderer, G. International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009

[16] IEC 61784, Industrial communication networks.

[17] EMC2 D8.4 – Mixed Criticality applications for FCS Scenario and Platform System Design Requirements (final version)

[18] Laprie, Jean-Claude. "Dependable computing: Concepts, limits, challenges." Special Issue of the 25th International Symposium On Fault-Tolerant Computing. 1995.

[19] EMC2 WP1 Technical Requirements,
FileName: EMC2_WP_ALL_REQ_INCL_REFERENCES_v10.xlsx
EMDesk: https://emdesk.eu/shared/56e2a443c60e1-51f9ebdbd9611f30615c4e1788c4c773

9. Appendix A

9.1 T1.6 Internal Report

Additional information about safety and fault-tolerance:
FileName: EMC2_T1p6_InternalReport_v1.0.doc
DocumentTitle: *T1.6 Internal Report of Safety and Fault tolerant concepts for SoA Embedded system Architectures*
EMDeskLink: **https://emdesk.eu/cms/?p=334&hash=alY3Q7Zm9sZGVyOzEwMTg2cmVkaXJ8**

9.2 Abbreviations

Table 1: Abbreviations

Abbreviation	Meaning
µC	Micro-Controller
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System Architecture
CAN	Controller Area Network
DRM	Design Reference Mission
dService contract	Dependable Service contract
dSOA	Dependable Service oriented Architecture
EBS	Enterprise Business Services
ECU	Electric Control Unit
EMC2	Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments
FlexRay	Automotive network communication protocol
HAZOP	Hazard and operability study
HW	HardWare
OS	Operating System
OSEK	Open Systems and their Interfaces for the Electronics in Motor Vehicles
POSIX	Portable Operating System Interface
PRA	Probabilistic risk assessment
QoS	Quality of Service
RTE	Run-Time Environment
RPC	Remote Procedure Calls
SD	Service Discovery
SOA	Service Oriented Architecture
SOMEIP	Scalable service-Oriented MiddlewarE over IP
SOTA	Software Over the Air
SW	SoftWare
WS	Web Service
WWW	World Wide Web
TMR	Triple Modular Redundancy