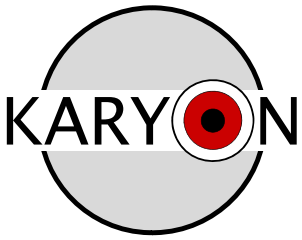

Kernel-based ARchitecture for safetY-critical cONtrol

KARYON

FP7-288195

D2.1 - First report on the KARYON architecture

Work Package	WP2		
Due Date	M9	Submission Date	2012-07-31
Main Author(s)	António Casimiro (FFCUL)		
Contributors	Rolf Johansson (SP) Kenneth Östberg (SP) Renato Librino (4SG) Jörg Kaiser (OVGU)		
Version	V1.0	Status	Final
Dissemination Level	PU	Nature	R
Keywords	Functional architecture, system architecture, system model		

Part of the Seventh

Framework Programme

Funded by the EC – DG INFSO

Version history

Rev	Date	Author	Comments
V0.1	2012-06-13	A. Casimiro (FFCUL)	First draft
V0.2	2012-07-15	A. Casimiro (FFCUL), R. Johansson (SP), K. Östberg (SP), R. Librino (4SG)	Update with new content in all sections
V0.3	2012-07-16	R. Librino (4SG)	Additional input from 4SG added
V0.4	2012-07-20	A. Casimiro (FFCUL)	Restructured some sections and highlighted needed inputs
V0.5	2012-07-24	A. Casimiro (FFCUL)	Improved some parts of the text and completed the global architectural view
V0.6	2012-07-26	A. Casimiro (FFCUL), R. Johansson (SP)	Added input from SP. Completed sections 2, 3 and 4.
V0.7	2012-07-30	A. Casimiro (FFCUL)	Final version with complete content and comments addressed
V1.0	2012-07-31	A. Casimiro (FFCUL)	Final review and delivery.

Glossary of Acronyms

ABS	Anti-lock Braking System
ADAS	Advanced Driver Assist Systems
ASIL	Automotive Safety Integrity Level
CAM	Co-operative Awareness Messages
COTS	Commercial Off-The-Shelf
DB	Database
DoW	Description of Work
Dx.y	Deliverable belonging to work package x, with serial number y
ETSI	European Telecommunications Standards Institute
HMI	Human Machine Interface
I2V	Infrastructure to Vehicle
ISO	International Organization for Standardization
ITS	Intelligent Transport Systems
KARYON	Kernel-based ARchitecture for safetY-critical cONtrol
LDM	Local Dynamic Map
LIDAR	Light Detection And Ranging
LoS	Level of Service
PICS	Protocol Implementation Conformance Statement
RADAR	Radio Detection And Ranging
RSU	Road Side Unit
SIL	Safety Integrity Level
TCB	Timely computing Base
TPM	Trusted Platform Module
TTCB	Trusted Timely Computing Base
TVRA	Threat, Vulnerability and Risk Analysis
Tx.y	Task belonging to work package x, with serial number y
V2I	Vwhicle to Infrastructure
V2V	Vehicle to Vehicle
V2X	Vehicle to Vehicle or to Infrastructure
WP	Work Package
WPx	Work Package with serial number x

Executive Summary

The main objectives of WP2 are to the definition the KARYON safety architecture, providing the guiding principles on how to structure a safe system in relation to assumed system and fault models. This will be done while taking into account that systems can be built from heterogeneous application components, where some components may provide higher levels of integrity than others, possibly due to being less complex or providing reduced functionality. The goal is to define a hybrid system architecture that integrates all these components in a way that it becomes possible to secure critical safety requirements while achieving higher levels of functionality. This is the first deliverable within this work package, providing a preliminary description of the KARYON architecture.

The work build on previous results achieved in WP1 and described in deliverable D1.1, in particular a set of requirements on the architecture. These requirements are considered and the deliverable provides an analysis of their implications on the architecture. Additionally, the notion of architectural hybridization is presented and explained in detail, since it is on the basis of the architectural work and solutions developed in the project.

The preliminary architecture described in this deliverable provides a high-level view on how a KARYON system must be structured, laying down the fundamental architectural blocks and describing their purpose and function, as well as the generic properties that they must enjoy. A data-oriented perspective of the architecture is also provided, which is useful to identify and to reason about the relevant interactions between architectural blocks. A discussion on how the presented architecture is appropriate to address the general requirements identified in WP1 is also provided.

The contributions of this deliverable are relevant and related to other activities. On the one hand, the fault and failure modes that are being addressed also in WP2 are intended to complement the architecture. They are addressed in another deliverable and only considered here whenever necessary for the presentation. On the other hand, the level of abstraction that is used to present the architecture is intentionally high, and independent from the implementation. However, there are implications on how the architectural blocks are implemented, and in this deliverable we include a discussion on these implications. They will be taken into account in the activities performed in WP3 and WP4.

Table of Contents

List of Figures

List of Tables

1. Introduction

The main objective of WP2, as stated in the KARYON Description of Work, consists in "*the definition of the KARYON safety architecture, providing the guiding principles on how to structure a safe system in relation to assumed system and fault models*".

KARYON focuses on the predictable and safe coordination of smart vehicles that autonomously cooperate and interact in an open and inherently uncertain environment. Although it is possible to exploit the cooperative functionality for the benefit of each vehicle's behaviour, with implicit gains to vehicles as a whole and to traffic, it becomes necessary to deal with the possibly negative impact of the uncertainties affecting communication and ultimately creating safety problems.

There is a whole body of knowledge on how to achieve safe systems, but in general the existing solutions and approaches are restrictive regarding the considered operational environments, excluding the sources of uncertainty or unpredictability right from the start and thus limiting the contexts in which the resulting systems can be used. Uncertainty can also be dealt with by making pessimistic worst case assumptions on bounds for the relevant variables. The consequence in this case is that system resources are over-dimensioned and hence the resulting systems are less efficient.

In KARYON we explore the concept of architectural hybridization (and corresponding hybrid system models), to define a generic architecture that accommodates both complex functions that might be subject to uncertainties, and simple, with well-defined behaviour functions, which are fundamental to elaborate on safety. The architecture will address the tension between these two different realms of operation, as needed to have benefits from a very complex cooperative control system, while ensuring that safety is preserved by means of the well-defined system component, a local safety kernel. In essence, architectural hybridization explicitly separates different functions or components of the system into different parts, where each part enjoys a specific set of properties (for instance, each part having different timeliness properties or different integrity levels with respect to some assumed failure modes). These heterogeneous properties will be reflected on the system model, and may be explored in the design of protocols and system solutions. On the implementation level, when allocating functions to specific resources, it will be necessary to ensure that the desired and expected properties will be effectively achieved.

In this deliverable we thus provide a preliminary description of the KARYON architecture, explaining the concept of architectural hybridization and how it is applied. More specifically, the deliverable provides functional and data-oriented views of the proposed KARYON architecture, including a description of the necessary functional components, of their properties and role within the hybrid architecture, and of the data and control flows that exist in a KARYON system. Furthermore, in the deliverable we also discuss how the proposed architecture addresses (and results from) the requirements defined in WP1, thus bridging the architectural work in WP2 with the work in WP1, and we study the implications of the proposed architectural solution on the design and implementation work that will be developed in other work packages, in particular in WP3 and WP4. In order to further test the implications and relations between the proposed abstract architecture and the more concrete system architectures that need to be defined when going to lower levels of abstraction, we instantiate the generic KARYON architecture to specifically considered cooperative functionalities in the automotive domain. We intend with such exercise to improve our knowledge and gain insights on how to possibly improve the generic architecture, which will be done in the continuation of the project.

In fact, given the preliminary nature of this deliverable, we expect to further refine the presented ideas, which will be described in the KARYON architecture deliverable, D2.3, in March 2013. Nevertheless, this deliverable sets the ground for the work to be carried out in WP3 and

particularly in WP4, namely the definition of the safety kernel. It also provides input to WP1, where work on the generalization of requirements and applicability of KARYON solutions to different application domains will be done. The work presented in this deliverable is also tightly related to the work on failure modes and semantics, which is being done in Task 2.2 and will be reported in deliverable D2.2.

2. Implications of the general requirements on the architecture

One of the results of the work performed in WP1, and reported in deliverable D1.1, was the definition of a set of general requirements on the architecture. These requirements were elaborated based on the defined automotive and avionics use cases, as well as on initial KARYON concepts. In this section we consider these requirements and provide a preliminary analysis of their implications on the architecture.

R.4.2.10

Each vehicle shall be able to perform several functionalities (services) simultaneously

Rationale: It is assumed that there are several functionalities of the vehicle of interest. This is the case for all vehicles of today, and also assumed in the vehicles we study in the use cases. This implies that when defining a KARYON system/architecture it cannot be enough only assuming to implement one single functionality. Much of the complexity making the solution general is that it should be able to handle all functionalities at the same time.

Implication: An architecture shall not be tailored for performing just one single functionality. Mechanisms and architectural patterns shall allow several functionalities to be taken care of simultaneously.

R.4.2.20

The set of functionalities shall be extendable

Rationale: This requirement is important for any architectural pattern to be exploitable for a real vehicle developer. We assume that incremental product development must be supported in such a way that the addition of one functionality should not require a completely new architecture.

Implication: The architecture pattern shall be so general that when adding one functionality, the same pattern shall still be valid. This shall hold even if the architecture instance is extended.

R.4.2.30

Each functionality shall be able to involve some sensing, actuating, and communication with other vehicles and infrastructure

Rationale: This requirement is a direct consequence of the use case criteria that we are looking at cooperative vehicles. The implication on the architecture is that for the realization of every functionality, sharing resources with actors outside the vehicles (other vehicles and infrastructure) shall be possible.

Implication: The architecture pattern shall deal with the inherent redundancy coming from a combination of local sensors and of communication with remote sensors. Taking advantage of inherent redundancy is key factor for reaching enough safety with a minimum cost of added redundancy.

R.4.2.40

Some resources for sensing, actuating and communication shall be able to be shared among several functionalities

Rationale: When adding a new functionality to a vehicle, it should be able to take advantage of the fact that some sensing and/or some actuating from other functionalities can also be used in the new one. A general KARYON architecture must give the possibility for several functionalities to share some resources.

Implication: The architecture pattern for realizing functionalities with elements shall be a many-to-many relation where:

- Each architectural element may be part of several functionalities
- Each functionality may be realized by several architectural elements

R.4.2.50

Each functionality shall always behave safely independently of the level of service

Rationale: If the available level of integrity becomes too low for the actual level of service, a transition to a lower level of service shall be done immediately (the time to initiate the transition shall be much shorter than the time for the transition itself).

Implication: The architecture shall be built by a proper combination of

- Components, having high enough integrity
- Redundancy patterns, lowering the requirements on integrity of components

R.4.2.60

Each functionality shall always operate in the highest possible level of service

Rationale: If the available level of integrity becomes high enough for a higher level of service than the actual one, a transition to a higher level of service shall be done immediately (the time to initiate the transition shall be much shorter than the time for the transition itself).

Implication: The architecture shall, for all functionalities at the time, enable a dynamic matching:

- Available level of integrity (from status of components)
- Required level of integrity (according to different levels of service)

R.4.2.70

A KARYON architecture shall be able to express on different levels of abstraction.

Rationale: This is to match a break-down of safety-requirements, and different phases in a safety standard reference life-cycle. .

Implication: The KARYON architecture is not just *one* view. It's important to represent the architectural pattern on several levels of abstraction. This enables separation of concerns, as different levels of abstraction have different concerns. The number of levels of abstraction shall be at least 4, to match the phases of the ISO26262 reference life cycle.

R.4.2.80

On each level of abstraction, and for each architectural element, the level of integrity shall be possible to express w.r.t. each applicable failure.

Rationale: This means a capability to express safety requirements having Safety Integrity Levels (SIL) and being allocable to any failure of any architectural element. This requirement implies that we need failure models of the architectural elements we use.

Implication: In a top-down methodology, the integrity levels identified in hazard analysis on the vehicle level shall be inherited to those architectural elements on analysis level w.r.t. corresponding failures. In a similar way, SIL w.r.t. failures on any level of abstraction shall be inherited to next level below, if no redundancy is introduced. Furthermore this implies that SIL be expressed as an attribute of safety constraint referencing a fault/failure model.

R.4.2.90

There shall be a known set of rules regarding how to determine the level of integrity for avoiding each possible resulting failure when composing architectural elements.

Rationale: This implies rules for SIL inheritance and for SIL decomposition (effects of redundancy).

Implication: If redundancy is introduced, instead of just inheritance, a lowering of SIL may be done according to applicable rules (e.g. ASIL decomposition in an automotive context).

R.4.2.100

There shall be a known set of rules regarding how to determine the level of integrity for avoiding each possible resulting output failure of an architectural element, given the integrity levels of avoiding the applicable input faults and internal faults.

Rationale: This implies a requirement on models for failure behaviour of all architectural elements.

Implication: For each architectural element on each level of abstraction, there is a need for a corresponding fault/error/failure model. These models include failure propagation behaviour.

R.4.2.110

There shall be known rules regarding how the amount of, and the quality of, relevant information determines the level of integrity for each relevant failure.

Rationale: This requirement asks for transformation rules from the “quality of information” domain to the “integrity level” domain. The former domain is what can be measured by the system itself and the latter domain is where the use case requirements are set. In order to understand when to go up and down in levels of service, such transformation rules have to be established that are applicable for the architecture and its elements.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine available integrity w.r.t. all relevant failures.

R.4.2.120

The amount of relevant information shall be measurable.

Rationale: There shall be a way for a KAYON system to dynamically extract what is needed to determine the available levels of integrity. Provided that the requirement on a transformation rule to determine the integrity level is fulfilled, then the amount of relevant information should be measurable by the system itself as an input to that transformation.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine available integrity w.r.t. all relevant failures.

R.4.2.130

The quality of relevant information shall be measurable.

Rationale: There shall be a way for a KAYON system to dynamically extract what is needed to determine the available levels of integrity. Given the requirement on a transformation rule to determine the integrity level is fulfilled, then the quality of relevant information should be measurable by the system itself as an input to that transformation.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine the available integrity w.r.t. all relevant failures.

3. Hybrid system models and architectures

In KARYON we exploit the concept of architectural hybridization in the definition of the KARYON architecture, in particular to realize the separation of the overall system in parts that have different properties. In this way, we are able to identify the components that constitute the safety kernel, which are in charge of guaranteeing that the intended functionality is provided in a safe way despite faults and uncertainties.

This section introduces the concept of hybrid system models and the corresponding architectural hybridization paradigm. To fully and clearly explain this concept, we start with an overview of the different and fundamental approaches for defining system models. We are essentially interested in showing the difference between models that assume homogeneous properties for the whole system, and hybrid models, which assume that system may enjoy different sets of properties. This is an important difference, with impact on how solutions are designed and on how the system will perform. We describe specific advantages of using hybrid system models in comparison to homogeneous ones.

However, simply assuming that a hybrid model is adequate to represent the real system is not enough. This must be reflected on the architecture and it is necessary to materialize the assumptions, ensuring that they hold in practice. This is why the architectural hybridization paradigm is essential, as it defines a set of principles for architecting the system and, in fact, enabling the construction of realistic hybrid systems. We thus elaborate on this in the following paragraphs, also providing some examples of such architecturally hybrid systems.

3.1 Hybrid system models

In a general sense, when designing a system or an application, or simply the solution for a given problem, it is necessary to clearly identify and specify a set of requirements for that system or problem, and a set of assumptions about the properties of the environment for which the problem is to be solved. While the set of requirements is what defines the problem, the set of assumptions has an implicit impact on the possible solutions, determining, for instance, their complexity. The set of assumptions is in fact a representation of the system in which the solution will be deployed, and thus constitutes the system model.

The system model provides an abstraction of the real system, allowing for the separation of concerns between the underlying system properties that the solution designer can take as granted, and how these properties are provided or enforced. Therefore, when we use the term system model we refer to an abstract representation of a real system, hiding details related to hardware, network and software components.

Abstracting is good, but it is important to ensure that the abstraction is accurate with respect to the reality it represents. There is an issue of assumption coverage [1111] that is relevant when the actual solution is deployed, that is, assumptions must hold with a high enough probability given a concrete system and environment. In essence, the right assumptions must be made. Additionally, the system model should be simple enough to be useful when designing some solution, but it should also be detailed enough to capture the essential characteristics of the system and allow better solutions to be defined.

Assumptions can be defined along several dimensions, depending on what is relevant for the problem at stake. For instance, in the distributed systems literature [8] a distributed system model includes assumptions about: (i) failures, (ii) synchrony, (iii) network topology and (iv) message buffering. In KARYON we are essentially interested in modelling failures, which is crucial to reason about safety. We do this essentially in the scope of Task 2.2, the task concerned with the definition of failure models and failure semantics. Furthermore, since we

consider systems that interact with their physical environment, the temporal and timeliness aspects are also important, and thus it is relevant to devote attention to synchrony assumptions, defined by a synchrony model. In fact, fault assumptions can be related and may depend on synchrony assumptions, in the sense that if some synchrony is assumed, then it might be necessary to also assume timing faults on the fault model. The same can be said regarding security-related assumptions and the implications of those assumptions on the fault model.

There is a wealth of knowledge on the definition of homogeneous system models, and on their use in the definition of algorithmic solutions, architectures and systems. For instance, when considering the synchrony dimension, the two well-know models of synchrony that have been traditionally used are the synchronous [7] and the asynchronous [6] models. The shortcomings of these homogeneous models are clear when dealing with problems where it is necessary to reconcile predictability with uncertainty [15], such as we do in KARYON.

Recalling the KARYON main objective, which is to provide system solutions for predictable and safe coordination of smart vehicles that autonomously cooperate and interact in an open and inherently uncertain environment, the need for reconciling predictability with uncertainty is evident. Let us reason again in terms of the synchrony dimension. Should we consider the asynchronous system model, we would have no way of addressing timeliness requirements and providing timeliness guarantees for the behaviour of the developed systems. In essence, ensuring functional safety would not be possible, given that even simple hazards require some (temporally) bounded system reaction, something that cannot be handled when considering an asynchronous model. On the other hand, despite the technology improvements in computing and communication, we should also not use a synchronous model in an unrestricted manner. For example, to deal with uncertain wireless communication delays, a synchronous model would either postulate a very high bound for the message delivery delay, which could be unacceptable for performance, or else, by postulating a lower bound the risk of violating the assumption could be too high and unacceptable.

It is possible to move away from the extreme sides of the spectrum of choices (be it about synchrony, security, integrity, or others), defining intermediate models for whatever considered dimension. For instance, in the synchrony domain there exist models of partial synchrony, such as the Partially Synchronous model [5] or the Timed Asynchronous model [4]. In these cases, synchrony is assumed to vary over time and, in this sense, is not an invariant property. However, since the property is assumed to be common to the entire system, the synchrony model is still homogeneous in the space dimension.

In contrast with homogeneous models, a hybrid system model allows possibly several stripes of the assumption spectrum to be represented, exploiting the space dimension. Then, provided it is possible to find a mapping of such hybrid models onto (correspondingly hybrid) architectural models that reflect reality (the networking and computational environment), it will be possible to exploit the increased expressiveness of the hybrid models to design improved solutions and, in particular, to address the conflicting goals of predictability and uncertainty.

In essence, hybrid system models represent systems in which different parts have different properties and can rely on different sets of assumptions (e.g., faults, synchronism). Interestingly, it is possible that some of these assumptions, applicable to some part of the system, lie in some intermediate point of the possible spectrum. Therefore, hybrid models allow the best to be taken from both dimensions: different loci of the system may have different properties, and these properties may vary over time.

In theoretical and practical terms, hybrid models have a number of advantages when compared to homogeneous models, as explained in what follows (a detailed discussion can be found in [16], focusing in particular on synchrony models).

Hybrid systems models are:

- Expressive models with respect to reality— Real systems are not homogeneous. Whatever the dimension (synchrony, integrity, etc) they generally have components that enjoy different properties, because these components use and depend on different resources (e.g., hardware devices, networks). Homogeneous models simply cannot take advantage from this, being confined to use worst-case assumptions (e.g., the most severe failure mode, the weakest synchrony).
- Sound theoretical basis for crystal-clear proofs of correctness— By using a hybrid model, the heterogeneous properties of the different loci of the system (the space dimension) are by nature represented, and we are in consequence forced to explicitly make correctness assertions about each of these loci, and about the interfaces to one another. In contrast, in homogeneous models (and particularly if they make weak assumptions) designers are tempted to make implicit assumptions that are not explicit in the model, which may lead to problems ahead.
- Naturally supported by hybrid architectures— Sisters to hybrid systems models, hybrid architectures accommodate the existence of actual components or subsystems possessing different properties than the rest of the system. Hybrid models and architectures provide feasibility conditions for powerful abstractions which are to a large extent unimplementable on canonical (homogeneous) models: timely execution triggers (also known as watchdogs); secure signatures or highly reliable execution kernels. Hybrid models and architectures may drastically increase the usefulness and applicability of all these abstractions.
- Enablers of concepts for building totally new algorithms— A powerful yet simple concept behind the first experiments with hybrid models was: use the weakest possible model for the generic system; imagine that a “toolbox” of simple but stronger low-level services is available, locally accessible to processes (e.g., timely execution triggers; timely executed actions; trusted store); these local services can be distributed via alternative channels, to obtain further strength (e.g., synchronous channels; trusted global time; trusted binary agreement); devise algorithms which, by working between the two space-time realms, the generic and the enhanced subsystem containing the “toolbox”, achieve new properties (e.g., making an asynchronous process enjoy timely execution).

Having explained the concept of hybrid system models, and their advantages over homogeneous models, in the next we address the architectural hybridization principle, as a fundamental enabler of the concept.

3.2 Architectural hybridization

Hybrid modelling of distributed systems is the path to achieving incrementally stronger behaviour taking the best of two worlds: retaining essentially weak models (of integrity, synchrony, security, etc), with consequent benefits for correctness (since assumptions are hardly violated); allowing strong models to be considered, which are essential to fulfil predictability and safety needs.

Architectural hybridization was proposed as a new paradigm to architect modular systems, based on a few simple principles:

- Systems may have realms with different non-functional properties, such as synchronism, faulty behavior, quality-of-service, etc.
- The properties of each realm are obtained by construction of the subsystem(s) therein.

- These subsystems have well-defined encapsulation and interfaces through which the former properties manifest themselves.

As to the construction, architectural hybridization is an enabler of the construction of realistic hybrid systems. In fact, it is quite straightforward to build architecturally-hybrid systems, and we provide some examples below.

The first example is of a system with a watchdog subsystem. The watchdog is used to reset or restart the overall system when something wrong happens in the main part of the system, typically when the main system becomes slow or inactive. The watchdog is essentially a counter device, which has a register that is programmed with some value, and a counter register that is continuously incremented. When the value in the counter register equals the value in the programmable register, the watchdog activates the reset signal. The main system has to periodically reset the programmable register to a higher value, to avoid system resets. When the main system becomes slow or stops, this will be implicitly detected because the programmable register will not be reset on time. In this example, it is easy to see that the system has two different parts, and is thus architecturally hybrid: the main system, which is assumed to fail or to behave untimely, and the watchdog, which is assumed to behave correctly and timely. These are reasonable assumptions, because the watchdog is essentially independent from the main system and it is a much simple subsystem. This ensures that faults affecting the main system will not propagate to the watchdog, and due to its simplicity the probability of the watchdog failing on its own is much lower than the probability of failure of the main system. Interestingly, the resulting global system exhibits better properties than the main system alone: it will either behave in a timely way or it will restart. In any case, untimely behaviours have been ruled out and this may be a useful property in many situations, when a fail-stop behaviour is admissible.

A second example is of a system with a Timely Computing Base (TCB) [14]. In such a system there is a generic part, called payload part, which corresponds to the baseline system where application processes execute to provide the intended application functionality. Then there is a control part, called the TCB, which like the watchdog is a separate part, but which provides richer supporting services to the payload part, like timing failure detection and timely execution of critical functions. It must be noted that the services provided by the TCB are distributed services, which implies that hybridization is extended to the network architecture. Clearly, the TCB part must be implemented in such a way that it enjoys better properties (reliability and timeliness) than the payload part. Some construction principles like interposition (ensuring that accessing to critical resources cannot be made bypassing the control part) and shielding (the control part is protected from faults affecting timeliness) must be respected to make sure that the services can be provided with the expected properties. One implementation of a TCB was done using Real-Time Linux and two switched Ethernet networks [1], where one of the networks was used exclusively for the TCB, whose services were implemented as real-time tasks. In this system, the payload part was the normal Linux part, using the other network. Another example implementation of a TCB, in which a completely separate hardware platform was used for the TCB subsystem, is described in [10].

One final example is a system with a Trusted Timely Computing Base (TTCB), in which hybridization is used not only to achieve a timely subsystem, but also a trusted subsystem, capable of providing security-related services like trusted random number generation and trusted block agreement [3]. Although the TTCB described in [3] was also implemented in Real-time Linux with specific changes in the kernel to enforce security properties, other COTS trusted hardware, such as the Trusted Platform Module (TPM) [12], can be used to obtain tamperproofness. In fact, a TPM can be seen as a special subsystem with better (security) properties which, when used in a generic (unsecure) system, ends up forming an architecturally hybrid system.

As to the usefulness of architectural hybridization, and considering the previous examples, it is clear that the overall system will be improved by making it able to use the services of a better

component or a better subsystem. For instance, in the case of the TCB it is possible to perform timely actions despite the asynchrony of the payload system, or to detect and react in a timely way to possible delays occurring in the payload part. On the other hand, with a TTCB it is possible to drastically augment resilience to intrusions, making it possible to solve fundamental problems such as consensus in the presence of uncertain attacks and vulnerabilities [9]. Note that in homogeneous systems, where the same fault, synchrony or security model applies to the entire system, the only way to achieve the intended (e.g., synchrony, security) properties is by enforcing these properties in the entire system, which is typically an overkiller. With architectural hybridization, only the restricted part of the system that has the better properties needs to be constructed with the aim of achieving those properties, which is much easier. And still, the provided services will make it easier to solve many problems that would otherwise not be solvable.

In KARYON we intend to apply architectural hybridization to exploit the better properties of a restricted part of the system in the achievement of the desired safe behaviour. In the next section we provide a preliminary description of the KARYON generic architecture, in which it will be clear how this hybridization is applied. As to the construction, or instantiation and implementation of a specific exemplifying architecture, we do not provide specific details in this preliminary deliverable, leaving that to the final architecture deliverable and to other KARYON work packages.

4. Architecture

In this section we provide a preliminary description of the generic KARYON architecture, which is divided in two parts. The first one is intended to provide a functional view, introducing the considered functional components and explaining their role in the architecture. The second provides an information flow view, introducing the main data abstractions that we need to consider and explaining the data flows between the functional components.

Our objective at this stage of the project is to include in the architecture the fundamental building blocks that will be the basis of any KARYON system. This is in accordance with the work plan, in which it is expected that this initial architecture proposition serves for the work that needs to be done in the remaining work packages, and that, as a result of the interactions between work packages, there will be refinements on the KARYON architecture, to be reported in the final architecture deliverable. Furthermore, the need to propose a sufficiently generic architecture also stems from the requirements listed in Section 2.

It is expected that, based on this general architecture, further work will be done in other work packages to solve specific problems implied by this architecture. Therefore, in the final part of this section we identify some of these specific problems.

4.1 Functional view

The architecture that we will be describing in this section was defined while having in mind the requirements identified in WP1, and taking into account the fundamental idea of applying architectural hybridization. We present the architecture as we understand it now, after a few iterations to accommodate diverse views and contributions. However, we try to provide some structuring ideas and abstractions which we hope will facilitate the understanding of the proposed architectural view.

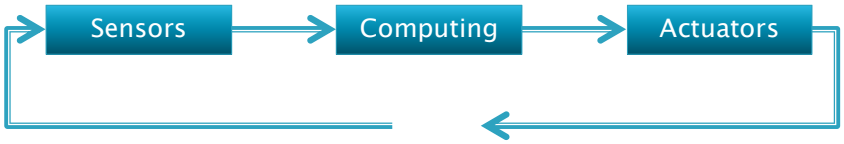

Figure 1: Basic control loop.

Figure 1 provides a high-level abstraction of a basic control system, which involves sensing, processing and actuation. This view abstracts the existing software and hardware components, as well as the communication channels connecting the components. In this view, there is an implicit feedback that develops through the environment. That is, through actuation it will be possible to change the behaviour of the controlled entity (which in KARYON is a vehicle), and this change will be perceived through the observation of physical variables that develop through the environment, like the ground relative speed or the distance to some physical object. It is well-known that it is easier to ensure a safe control according to the elaborated safety rules, in which some controlled variables are kept within desired bounds, when the system and the environment are well know and all dynamics can be predictably characterized. From a modelling perspective, this translates into considering synchronous models, well-defined failure modes, known event patterns, etc. And the solutions for safe control in such conditions are well known in the literature, implying a detailed system analysis at design time (i.e., statically), to prove (a priori) that the necessary safety conditions are met. Given that in KARYON we need and we want to deal with some level of uncertainty, this abstract architectural view must necessarily be enriched.

Firstly, let us slightly enrich the abstraction to make it explicit the fact that in a system there may exist, in fact, several sensors, computing elements and actuators. Let us also depict only the components, leaving the interactions for a later stage of analysis. We will call this basic system composed of sensing (Sense), computing (Compute) and actuation (Actuate) components, the nominal control system, as shown in Figure 2. The nominal system is the target system that we want to enrich, allowing it to provide improved functionality (with higher levels of service).

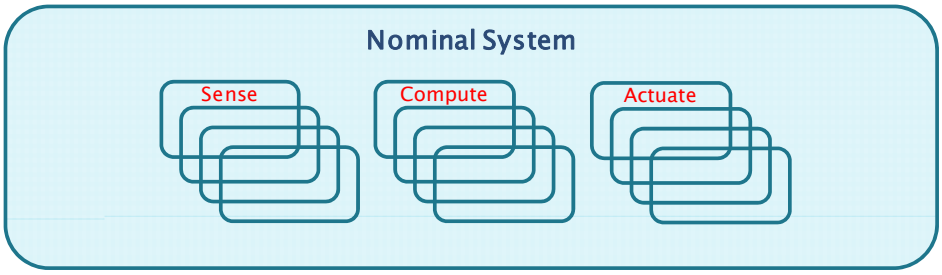

Figure 2: Nominal (control) system.

We consider that with the represented components it is only possible to provide local functionality, because none of the components supports the interaction with other nominal systems, which would be necessary to provide cooperative functionality. Therefore, in order to explicitly represent the need to communicate with other nominal systems, which is needed in KARYON, we add communication components to the nominal system model.

The new model is shown in Figure 4 and it now includes all the component types that we need. Sensing and actuation components implement the interface between the system and the environment. Sensing components consume information from the environment and produce information to the system. Actuation components, on the other hand, consume information from the system and produce information to the environment. Computing and communication components are just different in the sense that they consume and produce information from and to the system. Interestingly, this allows components to be modelled as objects providing both consuming (sensing) and producing (actuation) interfaces, and allows information to be modelled as events that flow from object to object. The Generic Events ARchitecture (GEAR) [2] provides the framework for reasoning in terms of *sentient objects*, which communicate through *generic events* and may be composed to create more complex sentient objects.

The communication components provide networking functionality, that is, they provide the means to connect a nominal system to other nominal systems. As described in GEAR, this communication is performed through *operational networks*, and is orthogonal to the sensing/actuation interfaces of the objects.

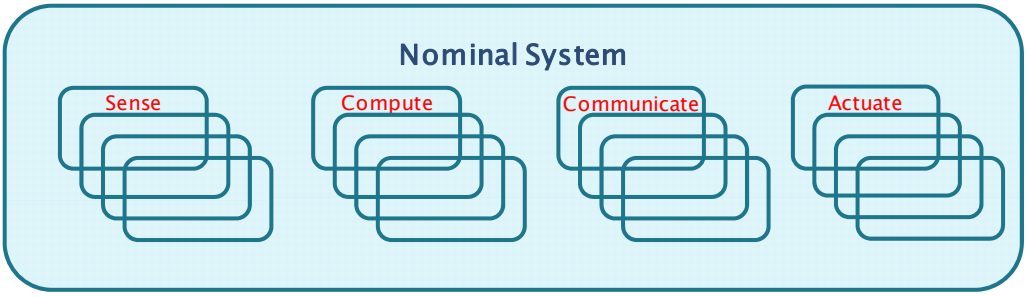

Figure 3: Nominal system for cooperative functionality.

It is important to say that the set of components that constitute the nominal system can be used in the provision of multiple functionalities. Adding a new functionality can thus be done by reusing some of the existing components and, possibly, adding just a few new ones.

When considering the need to support cooperative functionalities, and when adding communication components to the nominal system, we are implicitly adding uncertainty, which

cannot be handled at design time. In fact, since communication will be essentially wireless, this implies that it will be hard and inappropriate to assume fixed upper bounds for communication latency and predictability in general. Again from a modelling perspective, we are moving away from purely synchronous models and strong failure modes, which would allow us to use the well-known techniques for building real-time safety-critical systems. But this is what we proposed to do in KARYON, that is, be able to add complexity, richer components able to support improved functionality, while dealing with the increased uncertainty that this will bring to the system.

Besides supporting several functionalities, the objective is also to support the provision of different levels of service for each functionality. This means that the nominal system abstracted in Figure 3 moves even further away from a static system that is operating always within known bounds and providing a well-defined and fixed service. This also means that proving safety for the full range of admitted behaviours, conditions and faults becomes more difficult to do.

Recalling what we said in Section 3, we face the problem of reconciling uncertainty with the needed predictability. Assuming that we have a homogeneous nominal system enjoying synchronous behaviour is clearly not appropriate, because the assumed bounds would have to be too high. But we need some guaranteed behaviour in order to satisfy the safety needs. We need to add the needed predictability to the nominal system and the ability to adapt the nominal system in run time. This brings us to the proposed KARYON system architecture, as shown in Figure 4.

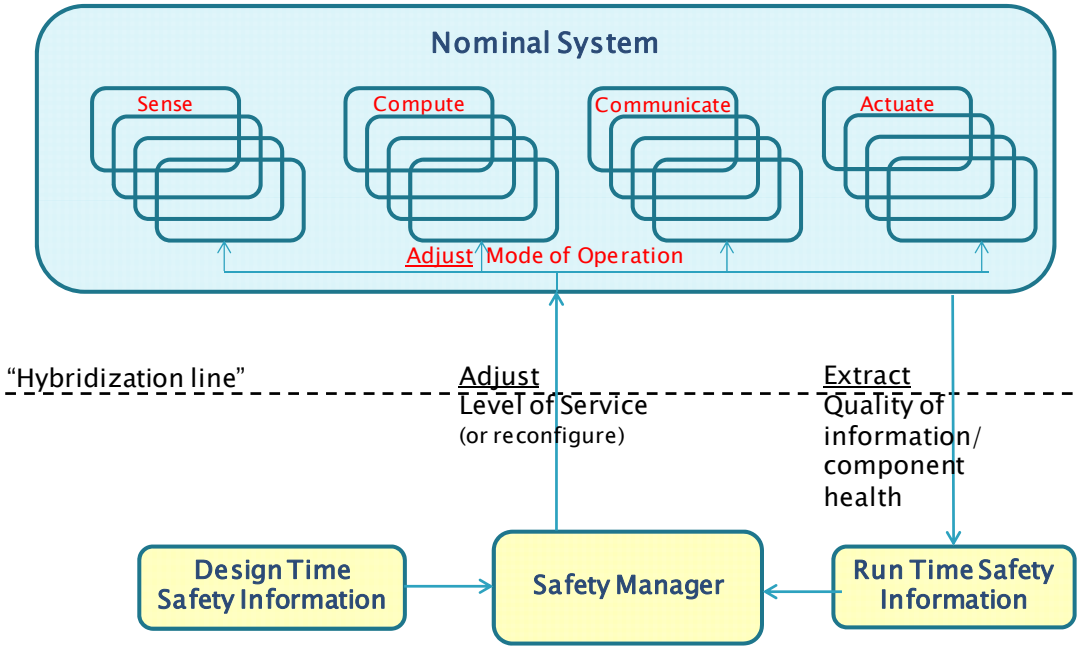

Figure 4: KARYON functional architecture.

In addition to the nominal system we add a *Safety Manager* component and associated *Design Time Safety Information* and *Run Time Safety Information* components. They constitute what we have been generically referring to as the safety kernel. We also highlight the separation between these components and the nominal system by means of the Hybridization line. This makes clear the need to assume that components in the different parts of the system enjoy different properties, and reflects the use of a hybrid system model and the application of the architectural hybridization paradigm. The following sections explain this architecture in more detail, addressing the underlying hybrid modelling approach, the functional description of the components, and the overall system behaviour in order to adjust the level of service of each functionality to match the available conditions and meet the safety objectives.

4.1.1 Applying the architectural hybridization paradigm

The hybridization line separates the system in two parts, denoting the application of the architectural hybridization paradigm. This allows making explicit the fact that different properties are assumed for each of the parts above and below the line.

Above the line, we have the nominal system that provides the intended functionality (or several ones), and has a diversity of components that may be combined and configured in different ways to provide a variety of levels of service for each functionality. It is not possible to prove at design time that the functionality will be safe for an arbitrary level of service independently of the anticipated conditions and faults. That is, some functionality provided with a certain level of service might not be safe if the conditions degrade, namely when there are failures affecting some components and leading to degraded data quality. However, it must be possible to statically prove that given some conditions the functionality will be safe in a certain level of service. Therefore, for the functionality to always be safe it is necessary to make sure that the level of service will be adjusted in run time to meet the observed conditions.

Above the hybridization line it is possible to explicitly accept weaker fault and synchrony models, which can be satisfied with less expensive resources, also allowing the use of a wider range of technologies (e.g., wireless networks, soft real-time schedulers) that are compatible with those weaker assumptions. The system will be dynamic and adapt in response to faults and to the available integrity level.

Below the line the system will be static. All the functional components must be statically proven to provide safe functionality independently of the anticipated faults. This means that these components, which constitute the safety kernel, will always operate correctly with respect to the assumed system and fault model for this part of the system.

However, it must be noted that it is also necessary to statically prove that the system will provide safe function for at least one level of service for each functionality. Therefore, the set of nominal system components that are necessary to provide this (lowest) level of service are, from a system modelling perspective, below the hybridization line, as shown in Figure 5.

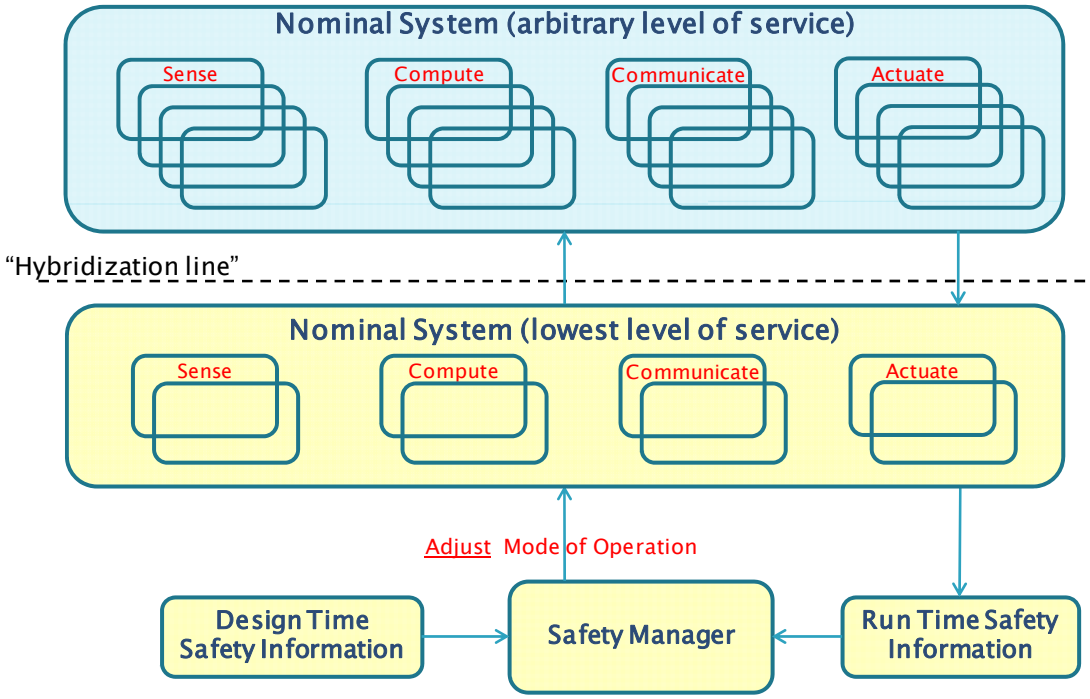

Figure 5: Separation of components according to hybrid system model.

The reader should be aware that in this figure we just aim at clarifying the distinction between what needs to be statically proven safe (below the line) or not. It does not mean that we must have strictly different components above and below the line. In fact, a single component might enjoy different properties (that is, be itself hybrid and on both sides of the line) depending on how it is configured at a certain moment and hence on the level of service it is expected that it provide.

This figure shows that we have one mode of operation, provided by the components below the hybridization line, which can be statically proven safe just as if we didn't consider the introduction of uncertainty and additional improved levels of service. If this was the only mode of operation the safety manager would not be needed, because this nominal system (when operating in this mode) is proved to provide safe functionality with respect to the considered hazards. But when adding functional components (above the hybridization line) or modes of operation for the existing components, the safety manager becomes fundamental to manage the mode of operation, allowing the system to switch between levels of service depending on the observed run time safety information. Clearly, it must be statically proven that the operation of the safety manager and its associated components will be safe, that is, the safety manager will issue safe management decisions. And there will be a set of components above the hybridization line that exhibits the reliability and synchrony properties that guarantee a functionality which is in compliance with the safety rules enforced by the safety manager.

4.1.2 Functional description of components

In contrast with the baseline model of a control system, shown in Figure 2 or in Figure 3, in Figure 4 it becomes clear that there is an additional control loop, in which the nominal system is being controlled by the safety kernel, more specifically by the safety manager component. In the execution of this control loop it is necessary to use design time and run time safety information.

4.1.2.1 Run Time Safety Information

According to the considered hybrid system model, we allow for some faulty behaviour of nominal system components. This uncertainty can be both in the time and in the value domain, being reflected on the validity of data that flows from one system component to the other. It results from the weaker synchrony and fault models that we can assume for these components. For instance, a sensor that may fail in several different ways will produce sensor data with varying validity levels, depending on the concrete faults that may have occurred and their direct impact on the sensor data values. And a communication link experiencing interferences may omit or delay the delivery of some messages, which will degrade the validity of data and the accuracy of the local view of the environment.

A crucial aspect of the proposed architecture is that instead of requiring the enforcement of some data validity or integrity levels, it just requires awareness about the validity of the data flowing in the system. From a more practical perspective (which will be discussed further ahead), we exchange the need for mechanisms to secure some predictable behaviour, by mechanisms to monitor the behaviour, detect faults, derive the validity of data and, in essence, be able to reason about safety. The set of collected information is represented in the architecture by the *Run Time Safety Information* component, which also abstracts the concrete mechanisms that must be put in place to do this information collection.

It should be noted that it is possible to collect different kinds of information that may serve to derive the validity of data or directly reason about safety. In fact, it may be possible that some of this data directly reports on the health of components, explicitly providing indications about the occurrence of faults affecting the component behaviour. For instance, it may be possible to know that some component crashed, simply stopping producing information. One open question, which is being addressed in the project, is whether it is easier or better to reason in

terms of this failure condition than to derive some validity for the data produced by the component (which in this case is not being produced).

Finally, we also note that knowledge about the context, meaning the physical surrounding environment, is usually important to reason about safety when considering vehicles that move and interact with this physical environment. This means that not only the validity of data is important for safety, but data itself may be important (if this data describes the physical context). We also do not restrict, at this stage, the possibility of including this context information as part of the run time safety information. However, since this data is in principle to be made available to the service itself, changes in the context can be reflected on how the function is performed rather than on the level of service. The best approach is still an open question.

4.1.2.2 Design Time Safety Information

The design time safety information consists of sets of safety rules establishing the conditions for functional safety assurance in each level of service. A certain functionality will only be safe in a given level of service (above the lower one), if the associated set of safety rules are satisfied at run time. This necessarily depends on the validity of data, and implicitly on the integrity of components, on faults and possibly on the physical context. Therefore, in order to evaluate safety it is necessary to have both the set of safety rules and the collected run time safety information.

Note that when a function is provided in the lowest level of service it is not necessary to verify if safety rules are met, because this has been done at design time.

For each level of service there is an associated set of safety rules. It must be proven at design time that if the safety rules are met, then the function will be safe in this level of service. The same has to be done for all levels of service and all sets of safety rules. However, it is not necessary to prove that these conditions will be met at run time. In fact, this is what distinguishes arbitrary levels of service from the lowest level of service. For the latter it is necessary to prove that (1) the function will be safe if safety rules are met and (2) safety rules will be met at run time. The corollary is that the function will be safe in the lowest level of service.

At run time, what needs to be done is to compare the current state of the system (conveyed by the run time safety information) with the safety rules for the current level of service. This is included in the tasks of the safety manager component.

4.1.2.3 Safety Manager

The role of the safety manager is to control the mode of operation of the nominal system components and hence adjust the level of service of each function. In order to do that, the safety manager needs to know the actual state of the nominal system, which is provided by the run time safety information. Then, given this state and given the safety rules provided as design time safety information, it decides whether the current level of service can be kept, or if the conditions determine a change of the level of service (either to a lower or to a higher one). This basic behaviour is illustrated in Figure 6.

The safety manager is permanently evaluating rules and determining possible adjustments of the level of service for some functionality. At this level of abstraction we are not specifying how this permanent evaluation is performed, but we anticipate that it will have to be done periodically, and with a known period. The latter is fundamental to prove that the safety manager will behave safely (as much as required).

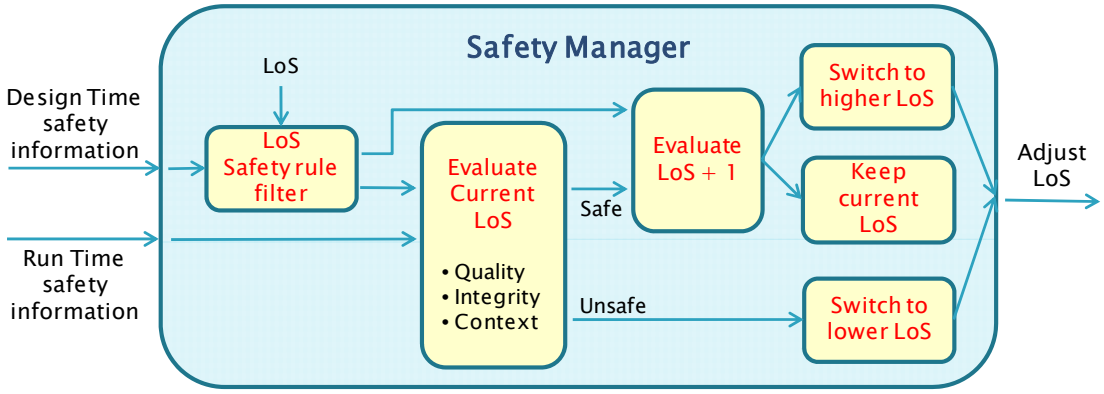

Figure 6: Safety manager basic behaviour.

The set of safety rules (for some functionality) that is used to evaluate safety is determined by the current level of service. If some rule is not satisfied given the available run time information, it becomes necessary to change the operation mode of nominal components, to force a switch to a lower level of service. Level of service changes are done on step at a time. That is, it suffices to determine that some safety rule is not met to trigger a change – it is not necessary to determine if there is some even lower level of service that would be more adequate to ensure safety. Given that the safety manager executes in a timely manner, it will be possible to know, in design time, how much time it may take to switch from the highest level of service to the lowest one, in which the functionality will assuredly be safe. Therefore, the safety rules that allow some high level of service to be provided, are derived (at design time) taking into account this upper bound on the time to switch again to a safe level of service.

It is possible that safety rules evaluate positively (actually this should be the normal situation), meaning that currently observed data validity, component integrity and context are good enough to keep providing the functionality with at least the current level of service. However, it might be possible that run time conditions actually allow providing a higher level of service for the functionality, not just keeping the current one. Therefore, if the evaluation of safety rules is positive, it is necessary to evaluate a new set of safety rules, for the next (higher) level of service. If they evaluate positively, then it is possible to switch to a higher level of service. Otherwise, nothing needs to be done.

The specific solutions to trigger reconfigurations or simple adjustments of the nominal system components are not fixed at this architectural level. We anticipate, however, that there will be pre-defined configurations and pre-defined reconfiguration plans, including the definition of all steps needed to execute these reconfigurations. A reconfiguration engine may be able to implement these plans, by command of the safety manager.

4.1.3 Levels of Service in the nominal system

Each functionality of the nominal system can be provided with several levels of service. The highest possible level of service should always be provided, which is determined by the integrity of data and system components that is available in run time. As mentioned in the previous section, the safety manager is able to determine if it is possible to switch to a higher level of service, allowing the highest possible level of service to be eventually reached. The architecture does not restrict the number of possible levels of service. This number will depend on the specific functionality and on other issues. For instance, it may simply result from a design decision, but it may also result from the hazard analysis and the identified risks to safety, which may require some specific levels of service to be considered in order to mitigate these hazards.

Typically, switching to a lower level of service is necessary when the integrity of data is not sufficient to raise the necessary certainty that the function can be safe while providing the

current level of service. For instance, if a sensor is not being able to accurately measure the distance to some vehicle in front of it (which might be detected by the sensor itself, by some external failure detection mechanisms, because there was a sudden variation of the value, or because an inconsistency is detected between the value reported by the sensor and the same value reported by another sensor or, say, the front vehicle's rear sensor), this will reflect on the validity of the sensor data, and ultimately on the ability of keeping the same level of service for a functionality to which this value is important. To preserve the required safety integrity level, it will be necessary to switch to a lower level of service, in which the available data validity will be sufficient to prevent any hazardous situation to develop.

We consider that there is a lowest level of service, and that in this lowest level of service the function is ensured to always be safe. This means that in this level of service, no hazardous situation can occur which would affect the required safety integrity level. This is possible because in such lower level of service there are lower resource requirements, fewer dependencies on nominal components and, in general, a reduced exposure to the possible hazards. In essence, the potential effect of hazards is discarded.

4.2 Information flow view

The description provided so far was essentially focused on the components and their functions, explaining why they need to be located above or below the hybridization line. Now we pay more attention to the interactions between the components, providing a data-oriented perspective of the KARYON architecture.

Central to this view is the notion that we have two fundamentally different kinds of data. On the one hand, there is application or service related data, which is necessary for the provisioning of the intended cooperative functionality. Considering the basic model of a control system presented in Figure 2, this is the data that flows from the environment through the sensors, computing components and actuators, back to the environment. In the absence of relevant risks, this basic control model would be enough and there would be no need to consider any other kind of data. However, we also need to consider safety related data, which is necessary for ensuring functional safety. In our case, since we consider several possible levels of service for each functionality, in order to ensure acceptable risks for each level of service we must continuously evaluate if the integrity of the components is the needed one. This integrity is reflected on the quality of service related data, and therefore we reason about safety using this quality information.

These two different kinds of data are shown in Figure 7 as "Service data" and "Quality data", and are included in the Run Time Information Database, which is just an abstraction to represent all run time produced data. Static information, on the other hand, is abstracted by the Design Time Safety Information Database, which specifically contains the safety rules derived in design time.

In addition to the information databases, in the figure we represent the components of the nominal control system and the safety manager. This data centric view is necessarily very abstract, in accordance with the functional view presented earlier. We do not consider concrete functionalities, nor concrete nominal system component, so it does not make sense to define specific flows between these components. What is relevant is that all the information (service and quality data) produced by sensing, communication and computing components constitutes run time information that may be required by other components, and should be made available to them. There may be several possible approaches to implement the communication between each component and to make run time information available, but the common denominator between all of them is that they should allow components to share information to all other interested components. Therefore, the idea that there is an abstract common information

repository, perfectly serves to represent this requirement. In the figure we represent all the data flows, irrespectively of their nature. In what follows, we provide more detailed views and explanations of each data flow.

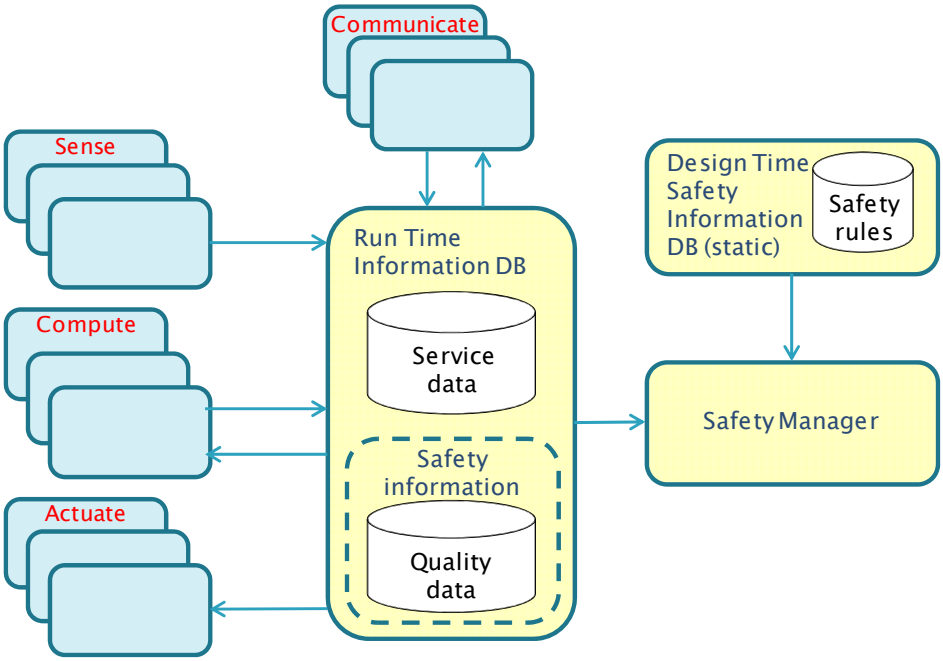

Figure 7: KARYON architecture (data centric view).

The flow of service data is just like the flow of data in a typical control system. This is clearly visible in Figure 8, in which only the service data flow and the relevant components are represented.

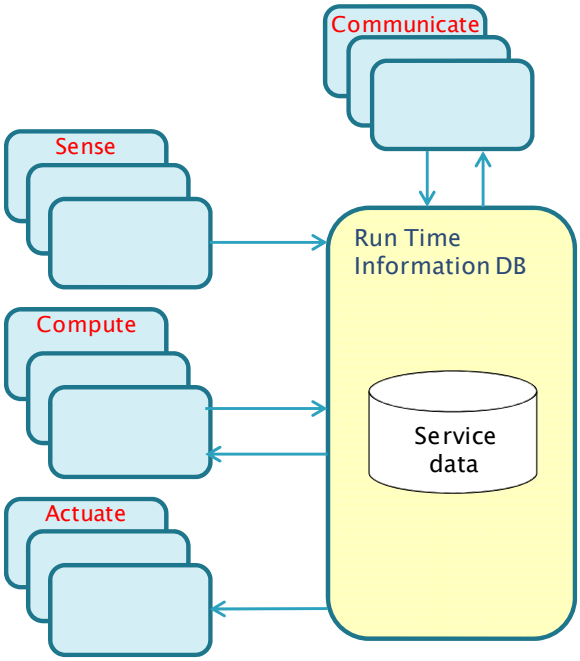

Figure 8: Service data flow.

Data is gathered by sensing components from the environment and by communication components from operational networks. These components then provide the collected information to computing components (through the Run Time Information Database), which process this information and produce new service data that may be either consumed by other

computing components, by communication or by actuation components. Communication components send this information through operational networks, while actuation components use the information to actuate on the environment, thus closing the control loop.

The flow of quality data is different, although there may be some overlaps with the service data flow. In fact, it is possible that information on data quality is transmitted along with the corresponding service data, in which case the overlap is obvious. For example, a distance value produced by a distance sensor could have some attached quality value and both values could be made available simultaneously. But this cannot be generalized.

Differently from service data, which originates from the environment or from a network, quality information is generated by some computing element, be this element part of a nominal component (e.g., an intelligent sensor, a data fusion computing component) or be it a computing component on its own. Therefore, in general it is necessary to consider that quality data can originate and flow from sensing, computing and communication components. This is illustrated in Figure 9, which shows the specific quality data flow through the relevant components.

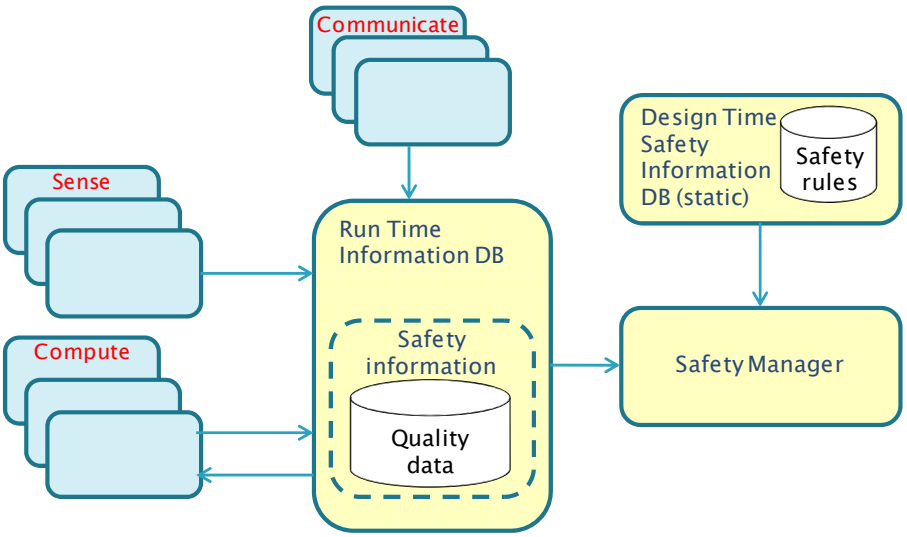

Figure 9: Quality data flow.

In the figure it is possible to observe that computing components can also consume quality information, for instance to derive the resulting quality for some produced service data. The main purpose of producing and gathering quality information is to make it available to the safety manager, as also represented.

It should be noted that some computing components may be devised to perform specific monitoring activities, like detecting crash and timing failures, which are important for evaluating the integrity of the run time system. This integrity-related information should be treated in some way, so that it is reflected in the quality of service data. In fact, this is a very important issue that has to be addressed in the context of the considered fault and failure models, which is the focus of work task 2.2. One of the KARYON objectives is precisely to define the relevant fault and failure models for sensors and the other system components, and understand how these faults and failures affect the service data quality. This is work in progress, whose preliminary results will be presented in Deliverable D2.2.

Another important issue concerning the quality data is that it should be trustworthy. In other words, it is useless for the safety manager to use information about the integrity of the system that might not be correct, or might be too imprecise. At design time, the confidence on this quality data must be established, and it must be assured by the implementation. Therefore, this is an issue that stems from the architectural work to the implementation work.

One final observation is that there is no output from the safety manager. This is due to the fact that the safety manager output cannot be considered service data, nor quality data. Another flow of control information must exist, which is directed from the safety manager to the relevant components that need to be reconfigured or readjusted. This flow is represented generically in Figure 4, because the concrete mechanisms and solutions to realize adjustments of the level of service pertain to a lower level of abstraction. This issue, which is also related to interfacing components in the two parts of the system, below and above the hybridization line, will be addressed in particular in work task 4.2.

4.3 Discussion on requirements fulfilment

In Section 2 the general requirements on the architecture are listed and elaborated to a certain extent. In this section follows a discussion on how the suggested architectural pattern fulfils these requirements.

R.4.2.10

Each vehicle shall be able to perform several functionalities (services) simultaneously

Rationale: It is assumed that there are several functionalities of the vehicle of interest. This is the case for all vehicles of today, and also assumed in the vehicles we study in the use cases. This implies that when defining a KARYON system/architecture it cannot be enough only assuming to implement one single functionality. Much of the complexity making the solution general is that it should be able to handle all functionalities at the same time.

Implication: An architecture shall not be tailored for performing just one single functionality. Mechanisms and architectural patterns shall allow several functionalities to be taken care of simultaneously.

Discussion: The elements “below the hybridization line” are not dependent on a specific functionality. Thus the requirement may be fulfilled.

R.4.2.20

The set of functionalities shall be extendable

Rationale: This requirement is important for any architectural pattern to be exploitable for a real vehicle developer. We assume that incremental product development must be supported in such a way that the addition of one functionality should not require a completely new architecture.

Implication: The architecture pattern shall be so general that when adding one functionality, the same pattern shall still be valid. This shall hold even if the architecture instance is extended.

Discussion: In the suggested pattern, the amount of functionalities realized “above the hybridization line” is arbitrary. Thus the requirement may be fulfilled.

R.4.2.30

Each functionality shall be able to involve some sensing, actuating, and communication with other vehicles and infrastructure

Rationale: This requirement is a direct consequence of the use case criteria that we are looking at cooperative vehicles. The implication on the architecture is that for the realization of every

functionality, sharing resources with actors outside the vehicles (other vehicles and infrastructure) shall be possible.

Implication: The architecture pattern shall deal with the inherent redundancy coming from a combination of local sensors and of communication with remote sensors. Taking advantage of inherent redundancy is key factor for reaching enough safety with a minimum cost of added redundancy.

Discussion: In the suggested pattern, all of sensing, actuating, and communication with other vehicles and infrastructure are possible and thus the requirement may be fulfilled.

R.4.2.40

Some resources for sensing, actuating and communication shall be able to be shared among several functionalities

Rationale: When adding a new functionality to a vehicle, it should be able to take advantage of the fact that some sensing and/or some actuating from other functionalities can also be used in the new one. A general KARYON architecture must give the possibility for several functionalities to share some resources.

Implication: The architecture pattern for realizing functionalities with elements shall be a many-to-many relation where:

- Each architectural element may be part of several functionalities
- Each functionality may be realized by several architectural elements

Discussion: In the suggested pattern, all of sensing, actuating, and communication with other vehicles and infrastructure are possible to implement as elements shared by several functionalities. This have to be further detailed, but so far the requirement may be fulfilled.

R.4.2.50

Each functionality shall always behave safely independently of the level of service

Rationale: If the available level of integrity becomes too low for the actual level of service, a transition to a lower level of service shall be done immediately (the time to initiate the transition shall be much shorter than the time for the transition itself).

Implication: The architecture shall be built by a proper combination of

- Components, having high enough integrity
- Redundancy patterns, lowering the requirements on integrity of components

Discussion: One implication on the suggested pattern is that everything “below the hybridization line” has to be implemented according to highest level of safety integrity. This part of the architecture has to be statically proven at design time to behave safely for all levels of service for all functionalities. Given that this can be shown, everything in the architecture “above the hybridization line” may take advantage of the possibilities of combining components having high enough integrity with redundancy patterns, lowering the requirements on integrity of components. Thus the requirement may be fulfilled.

R.4.2.60

Each functionality shall always operate in the highest possible level of service

Rationale: If the available level of integrity becomes high enough for a higher level of service than the actual one, a transition to a higher level of service shall be done immediately (the time to initiate the transition shall be much shorter than the time for the transition itself).

Implication: The architecture shall, for all functionalities at the time, enable a dynamic matching:

- Available level of integrity (from status of components)
- Required level of integrity (according to different levels of service)

Discussion: Given that the safety manager may observe the provided levels of safety integrity of all architectural elements and match this information with the required levels for the different levels of service, this requirement may be fulfilled.

R.4.2.70

A KARYON architecture shall be able to express on different levels of abstraction.

Rationale: This is to match a break-down of safety-requirements, and different phases in a safety standard reference life-cycle. .

Implication: The KARYON architecture is not just *one* view. It's important to represent the architectural pattern on several levels of abstraction. This enables separation of concerns, as different levels of abstraction have different concerns. The number of levels of abstraction shall be at least 4, to match the phases of the ISO26262 reference life cycle.

Discussion: The proposed architectural pattern may be described on several levels of abstraction, but the details, especially on the lower levels of details, have to be further elaborated.

R.4.2.80

On each level of abstraction, and for each architectural element, the level of integrity shall be possible to express w.r.t. each applicable failure.

Rationale: This means a capability to express safety requirements having Safety Integrity Levels (SIL) and being allocable to any failure of any architectural element. This requirement implies that we need failure models of the architectural elements we use.

Implication: In a top-down methodology, the integrity levels identified in hazard analysis on the vehicle level shall be inherited to those architectural elements on analysis level w.r.t. corresponding failures. In a similar way SIL w.r.t. failures on any level of abstraction shall be inherited to next level below, if no redundancy is introduced. Furthermore this implies that SIL be expressed as an attribute of safety constraint referencing a fault/failure model.

Discussion: For the architectural elements "below the hybridization line", it is assumed that any safety requirement allocated here may imply the highest level of safety integrity to be proven at design time. For all the elements "above the hybridization line", any safety requirement expressed on any level of abstraction, has to be according to an appropriate fault/failure model. It is assumed that work tasks 2.2 and 4.1 can identify this. If that holds, this requirement will be fulfilled.

R.4.2.90

There shall be a known set of rules regarding how to determine the level of integrity for avoiding each possible resulting failure when composing architectural elements.

Rationale: This implies rules for SIL inheritance and for SIL decomposition (effects of redundancy).

Implication: If redundancy is introduced, instead of just inheritance, a lowering of SIL may be done according to applicable rules (e.g. ASIL decomposition in an automotive context).

Discussion: In the proposed architectural pattern these rules are assumed to be identified at design time for every architecture instance and stored in the conceptual block called "Design Time Safety Information". The rules to be identified shall be consistent with the rules in the applicable safety standard regarding lowering required safety integrity level when introducing redundancy. Such rules are to be further investigated in other work tasks, especially in work task 4.1.

R.4.2.100

There shall be a known set of rules regarding how to determine the level of integrity for avoiding each possible resulting output failure of an architectural element, given the integrity levels of avoiding the applicable input faults and internal faults.

Rationale: This implies a requirement on models for failure behaviour of all architectural elements.

Implication: For each architectural element on each level of abstraction, there is a need for a corresponding fault/error/failure model. These models include failure propagation behaviour.

Discussion: In the proposed architectural pattern these rules are assumed to be identified at design time for every architectural element of the actual architecture instance and stored in the conceptual block called "Design Time Safety Information". Such rules about fault/failure models including rules for error propagation are to be further investigated in other work tasks, especially 2.2, but also 4.1.

R.4.2.110

There shall be known rules regarding how the amount of, and the quality of, relevant information determines the level of integrity for each relevant failure.

Rationale: This requirement asks for transformation rules from the "quality of information" domain to the "integrity level" domain. The previous domain is what can be measured by the system itself and the latter domain is where the use case requirements are set. In order to understand when to go up and down in levels of service, such transformation rules have to be established that are applicable for the architecture and its elements.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine available integrity w.r.t. all relevant failures.

Discussion: In the proposed architectural pattern the safety manager realizes the match of available integrity with required integrity. The necessary rules are assumed to be identified at design time for every architectural element of the actual architecture instance and stored in the conceptual block called "Design Time Safety Information". Such rules about transformation from fault/failure models to safety integrity levels are to be further investigated in other work tasks, especially 2.2, but also 4.1.

R.4.2.120

The amount of relevant information shall be measurable.

Rationale: There shall be a way for a KAYON system to dynamically extract what is needed to determine the available levels of integrity. Given the requirement on a transformation rule to determine the integrity level is fulfilled, then the amount of relevant information should be measurable by the system itself as an input to that transformation.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine available integrity w.r.t. all relevant failures.

Discussion: In the proposed architectural pattern it is assumed that "run time safety information" including measures on amount of relevant information can be extracted from the "architectural elements above the hybridization line" to the dedicated conceptual block below. How this may be done is to be further investigated in other work tasks, especially 4.2.

R.4.2.130

The quality of relevant information shall be measurable.

Rationale: There shall be a way for a KAYON system to dynamically extract what is needed to determine the available levels of integrity. Provided that the requirement on a transformation rule to determine the integrity level is fulfilled, then the quality of relevant information should be measurable by the system itself as an input to that transformation.

Implication: The architecture must encompass the necessary architectural elements to match available integrity with required integrity, and it shall be possible at runtime to determine the available integrity w.r.t. all relevant failures.

Discussion: In the proposed architectural pattern it is assumed that "run time safety information" including measures on quality of relevant information can be extracted from the "architectural elements above the hybridization line" to the dedicated conceptual block below. How this may be done is to be further investigated in other work tasks, especially 4.2.

5. Implications on services and mechanisms

In this section we identify and we provide a brief discussion of a set of issues that are implied by the KARYON architecture described in Section 4. These issues will be addressed as part of the forthcoming activities in the project, namely in the scope of WP3 and WP4. The work being done in work task T2.2, on failure modes and semantics, is also relevant to address some of the listed issues.

Characterization of quality information

Given that we assume that part of the system can be affected by faults (described by considered failure modes), which will be reflected on the quality of data, one issue is that it is necessary to find adequate forms for characterizing and representing the "quality" of data.

This data is provided by sensors and by communication components, and represents the state of physical variables, like distance, speed, temperature, heading, etc. Therefore, when using a data value representing some of these physical variables, there is an error between this data value the real physical value. The error is affected both by faults and is changing over time, and this is why it is necessary to continually update the value, to prevent the error to become too large. Under controlled conditions it is possible to make sure that errors are bounded, and design solutions that will be correct, and will be functionally safe, for the assumed maximum error. In KARYON we want to keep track of the validity of data, which in some sense corresponds to be able to characterize this error at run time. This will require being able to integrate the assumed fault models, so that the occurrence of faults can be detected and can be reflected on the validity of data. Furthermore, it will be necessary to find a generic way to represent this validity through some quality metrics, which might be easily used along with some algebra to reflect changes in this quality along the processing flow within the system.

It is also important to note that since the quality of data depends on the passage of time, it is important to preserve information regarding the time at which some quality information might have been derived.

Finally, the existence of the Run Time Information Database abstraction might be useful to deal with this data quality issue: this database may provide the means for the separation of concerns between data producers and data consumers, where producers are expected to ensure some desired quality of the information stored in this database, and consumers expect this quality to be ensured.

Mapping between quality and integrity

While the aim is to be able to being able to derive the validity of sensor data, and assign some quality value, reasoning about safety has to be done by considering desired safety integrity levels with respect to the considered hazards. There is an issue of matching the available quality to the needed integrity, which needs to be addressed in the project. In fact, this is also highlighted by general requirements R.4.2.100 and R.4.2.110, and this will be investigated in work tasks 2.2 and 4.1.

Level of service management

Considering that all issues related to the characterization of the quality of information and its transformation into the integrity domain can be done, then the safety manager will be able to evaluate if the available integrity is sufficient to keep some specific level of service. If not, the level of service must be changed. One issue is how to perform this change.

The change is controlled by the safety manager, and may be viewed as a reconfiguration procedure, in which new configuration parameters must be set for the relevant components. This implies that components may have to behave according to a set of parameters that determine a mode of operation. It may also be possible to envisage other kinds of reconfiguration, implying

activation or deactivation of components, or implying rebinding of component connections, but this should be abstracted in the same way through the change of system parameters.

Adaptation timeliness

One important aspect, though, concerns the timeliness of the reconfiguration actions.

The timeliness constraints are fundamental as they dictate the maximum amount of time that will take to complete a mode change. If one adds the maximum amount of time that it takes to collect integrity information (that is, the collection period) and check if safety rules are being satisfied, the resulting sum will provide a bound on the maximum amount of time that it will take to accomplish some needed adaptation. This amount of time will have to be taken into consideration when defining the safety rules.

It is necessary that the devised solutions will satisfy the required timeliness constraints. This should in principle be feasible given that this is commanded by the safety manager, which is a predictable (and timely) component. In any case, the issue calls for careful attention in the definition and implementation of interfaces, to make sure that these timeliness constraints are secured.

Actuation safety

As mentioned in the project proposal, the safety kernel part of the system "*safeguards the control commands and checks them against the derived set of safety rules*". In abstract, this is necessary when nothing is assumed about what control commands can be consumed by actuation components, or in other words, when nothing can be assumed about the quality of data used for actuation. In this situation, even if the integrity of actuation components is evaluated to be sufficient with respect to safety rules, the resulting actuation could impair the safety of the provided functionality. In fact, it follows from the proposed architectural pattern that the data flow goes directly from computing components to actuation ones (through the Run Time Information Database), and hence this data can be affected by faults that are only detected later, when the data has been used in actuation.

Therefore, it is necessary to either establish (and enforce) some quality/integrity level for the data that is used in actuation, which depends on the fault models that are assumed for the components producing this information, or else it is necessary to make sure that an interposition principle is followed in the implementation, so that the relevant data flowing to actuation components has to go through the safety kernel part of the system for prior validation.

6. Application to concrete functionalities

In this section we jump into a lower level of abstraction, performing a simple but important exercise: we consider specific functionalities in the automotive domain, and we describe possible architectural solutions, which hopefully can be seen as an instantiation of the generic architecture described in the previous sections. This exercise is important to reveal possible fragilities of the generic architecture (e.g. functional blocks or interactions that might not be adequate when we try to instantiate them), which will take us back to the drawing board in order to refine the architecture, to achieve the final version that will be provided in deliverable D2.3.

The automotive functionalities considered in KARYON are aimed, in particular, at road safety and traffic efficiency. These objectives are pursued by means of co-operative driving applications, in which traffic information is transmitted by the infrastructure and/or by other vehicles, and by means of Advanced Driver Assist Systems (ADAS), based on-board sensors that control the vehicle dynamics.

The set of requirements for these applications have been provided in D1.2, and are just briefly summarized here. We note that these requirements have been derived from the more significant standards relevant to KARYON and pointed by ETSI, which has been committed by the European Commission to prepare the standards for the European intelligent transport system. These requirements are therefore the basis of the KARYON automotive functionalities, which, in the following paragraphs, are described. For each functionality a functional architecture has been defined using the general structure of a nominal system, previously shown in Figure 2.

Starting from this structure and from the additional functions introduced to meet the requirements addressing complementary aspects, and also taking into account the results of the preliminary hazard analysis reported in D1.1, a general functional architecture for the above automotive applications has been identified. This architecture, which is intentionally very simplified in order to avoid the risk to stick to specific implementations, is proposed as a basis for a more accurate hazard analysis and risk assessment, and also to introduce the concepts that are emerging regarding the level-of-service approach and the management of functional safety by means of the KARYON elements.

6.1 Requirements from Automotive Standards

ETSI EN 302 665 Intelligent Transport Systems (ITS); Communications Architecture

- KARYON should assume the availability of the following infrastructures and services:
  - V2V communication: ITS-G5, 60 GHz, IR
  - Roadside stations: V2I and I2V ITS-G5
  - Collision Risk Warning RSU[1] (Road Side Unit)
- KARYON should assume that vehicles are equipped with 77 GHz RADAR or/and LIDAR systems
- KARYON should assume that the cooperative vehicles communicate with the ITS, whose architecture complies with the specifications of ETSI EN 302 665
- KARYON should assume that the facilities provided by ITS, in particular Local Dynamic Map (LDM) and support for relevance checking, are available and used to perform the required functionalities envisaged in KARYON use cases

- KARYON should assume that the support provided by ITS stations to manage Cooperative Awareness Messages is provided by ITS
- KARYON should adopt one of the (informative) onboard communication architectures provided by the standard.

ETSI TR 102 638 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions

- No additional requirements can be derived, because some of the most significant services defined by the standards are already defined in the previous chapter of D1.2.

ETSI TS 102 637-2 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service

- KARYON should include the Cooperative Awareness Messages to identify the global status of the surrounding environment

  For different ITS station types, the mandatory, situational mandatory and optional content of following tagged data:

  - Vehicle type – Public vehicle type
  - Light bar in use
  - Sirene in use
  - Emergency response type
  - Station length – Confidence of the station length
  - Station width – Confidence of the station width
  - Vehicle speed – Confidence of the vehicle speed
  - Longitudinal acceleration – Confidence of the longitudinal acceleration
  - Yaw rate – Confidence of the yaw rate
  - Acceleration control
  - Exterior lights
  - Cause code
  - Ambient air temperature
  - Speed, other speed than vehicle speed
  - PT line description
  - Turn advice
  - Distance to stop line – Confidence of the distance to stop line
  - Schedule deviation
  - Traffic light priority
  - Door open
  - Data reference
  - Confidence ellipse of the position
  - Curvature
  - Curvature change
  - Confidence of the curvature
  - Wiper system front
  - Crash status
  - Heading confidence
  - Dangerous goods

- KARYON should comply with the timing specifications of the Co-operative Awareness Messages (CAMs)

  CAMs are generated by the CAM Management and passed to lower layers when any of following rules apply:

  - maximum time interval between CAM generations: 1 s;

- minimum time interval between CAM generations is 0,1 s. These rules are checked latest every 100 ms;
- generate CAM when absolute difference between current heading (towards North) and last CAM heading > 4°;
- generate CAM when distance between current position and last CAM position > 5 m;
- generate CAM when absolute difference between current speed and last CAM speed > 1 m / s;

The generation rules are checked every 100 ms.

Use Case	min Frequency (Hz)	min Latency (ms)
Emergency Vehicle Warning	10	100
Slow Vehicle Indication	2	100
Intersection Collision Warning	10	100
Motorcycle Approaching Indication	2	100
Collision Risk Warning	10	100
Speed Limits Notification	1 to 10	100
Traffic Light Optimal Speed Advisory	2	100

Table 1: Overview Use Cases based on CAM (source: ETSI).

ETSI TS 102 868-1 Intelligent Transport Systems (ITS); Testing; Conformance test specification for Co-operative Awareness Messages (CAM); Part 1: Test requirements and Protocol Implementation Conformance Statement (PICS) proforma

- No additional requirement can be derived for KARYON activities, at the highest requirement level, because the requirements identified from the analysis of the preceding standard cover also the present one.

ETSI TR 102 863 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and guidance on standardization

- LDM concept shall be considered to create a global status of the environment
- The reliability issues of ITS station shall be considered by KARYON and suitable measures shall be taken to avoid possible hazards.

ETSI TR 102 893 Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA)

- KARYON shall consider the hazards caused by malicious attacks
- The KARYON architecture shall include countermeasures to ensure security
- KARYON, if will not develop the countermeasures, shall define them as assumptions.

ETSI TR 102 862 V1.1.1 (2011-12) Intelligent Transport Systems (ITS); Performance Evaluation of Self-Organizing TDMA as Medium; Access Control Method Applied to ITS; Access Layer Part

- KARYON should compare the access methods reported in the standard with the one under investigation by Chalmers University

6.2 Functionalities

The automotive functions considered in KARYON can be grouped in three categories:

- Co-operative driving, based on the Vehicle to Vehicle and Vehicle to Infrastructure communication
- Advanced Driver Assist Systems (ADAS), based on sensors, that are on board vehicle
- Vehicle dynamics control

6.2.1 Co-operative driving

The functions of co-operative driving are mainly based on:

- Cooperative Awareness Basic Service
- Cooperative Automatic driving

and are listed in Table 2.

Cooperative Awareness Basic Service	A) Intersection collision warning
	B) Signal violation warning
	C) Lane Change Manoeuvre
	D) Co-operative adaptive cruise control D1) Emergency brake lights D2) Stationary vehicle warning
	E) Intersection management E1) Traffic light optimal speed advisory E2) Collision Risk Warning from RSU E2) Signal violation warning
Automatic driving	F) Co-operative vehicle-highway automation system (Platoon) F1) Co-operative side merging F2) Co-operative roundabout merging
	G) Intersection control

Table 2: ITS and co-operative driving functions relevant to KARYON

In the following, the functions to be considered in KARYON are briefly described and diagrams are sketched to show the more significant elements needed.

Cooperative awareness function

Cooperative awareness functionality regarding road safety is a warning service based on the information about the status of the neighboring vehicles and of the road conditions, intended to alert the driver and safely anticipate the needed maneuvers. The information provided is standardized by ETSI. The Human Machine Interface (HMI) includes haptic signal on the steering wheel. In Figure 10, V2I and V2V mean the information provided by the various external sources, according to standardized ITS services (e.g. Local Dynamic Maps or CAM messages).

Cooperative automatic driving function

Cooperative automatic driving includes many possible functions, based on the available information about the neighboring vehicles and ranging from only longitudinal control to the complete vehicle control including lateral control. In general and in the complete functionality, automatic driving does not require any driving action by the driver, but usually the driver performs a surveillance task and should be ready to take the control in the case of risky situations or whenever the road conditions do not allow automatic driving (e.g. in complex traffic scenarios). The main functions that compose cooperative driving are shown in Figure 11.

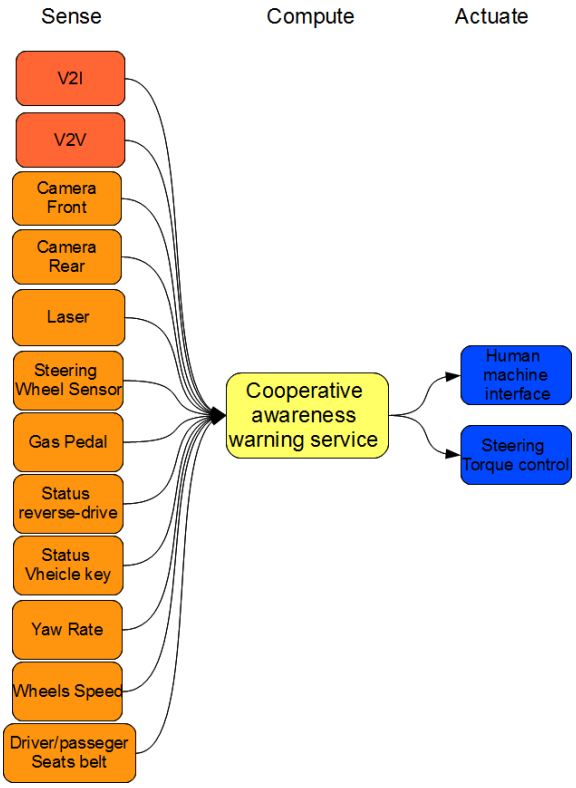

Figure 10: Functional architecture on board vehicle for cooperative awareness function.

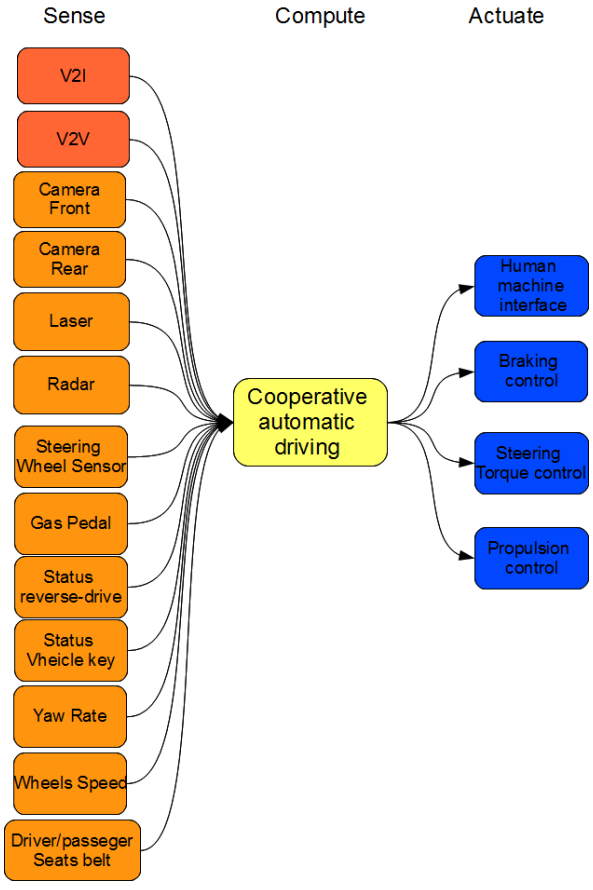

Figure 11: Functional architecture on board vehicle for cooperative automatic driving function.

6.2.2 Advanced Driver Assist function

Autonomous cruise control

It is an automatic cruise control that uses either a radar or laser sensor setup, with the support of a camera, allowing the vehicle to slow when approaching another vehicle ahead and accelerate again to the preset speed when traffic allows. Also in this case the functionality of the automatic cruise control does not require any driving action by the driver, but usually the driver performs a surveillance task and should be ready to take the control in the case of risky situations or in complex traffic scenarios. The functional architecture on board vehicle is shown in Figure 12.

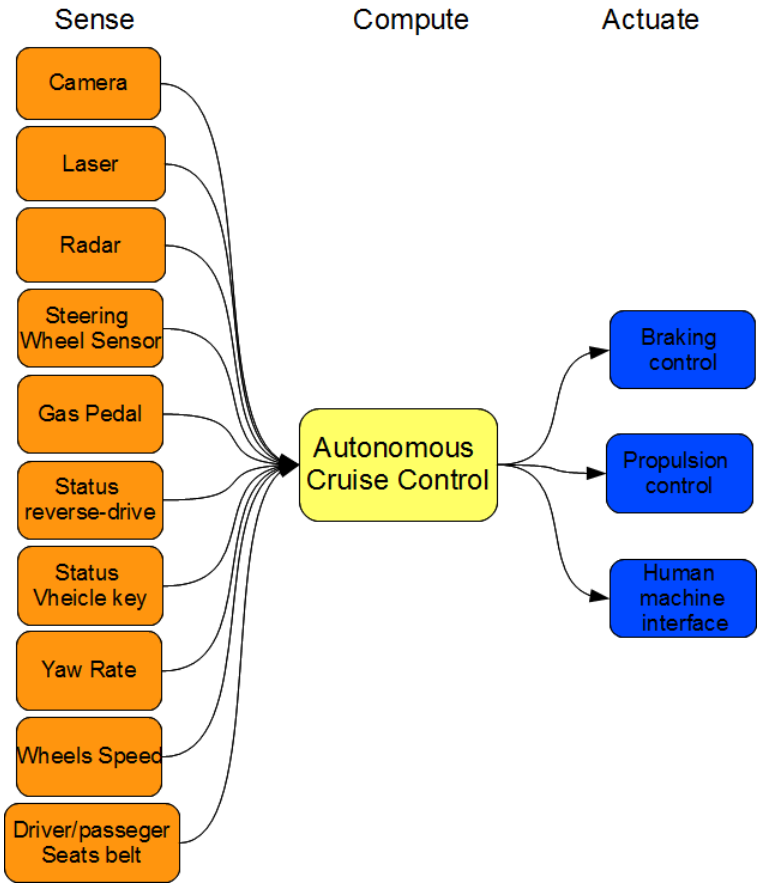

Figure 12: Functional architecture on board vehicle for autonomous cruise control.

Lane departure warning function

It is a mechanism to warn the driver when the vehicle begins to move out of its lane (unless a turn signal is on in that direction) on freeways and arterial roads. The warning to the driver can be performed also adding haptic feedback, directly on the steering wheel. The functional architecture on board vehicle is shown in Figure 13.

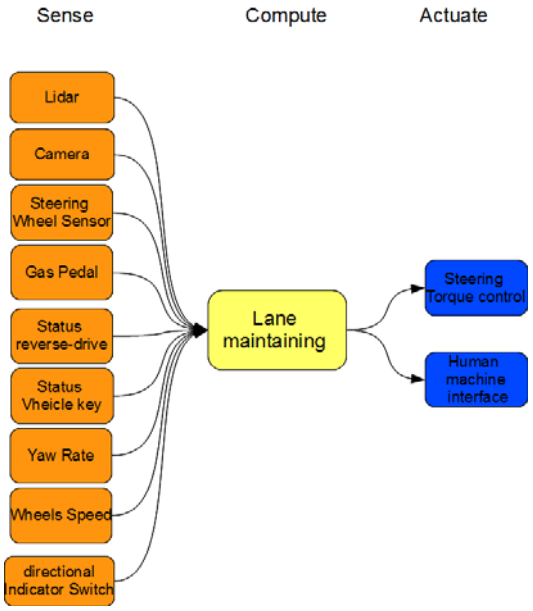

Figure 13: Functional architecture on board vehicle for lane departure warning function.

Collision avoidance function

This functionality is based on the detection of moving obstacles on the vehicle trajectory, by means of radar systems with an obstacle detection range of at least 150 m, offering brake assist support across a full range of speeds. In case of an imminent collision with an object in front, automatic braking support is triggered, helping to mitigate impact or avoid collision, the system triggers a warning to alert the driver. The warning can be audible, visual or haptic. If the driver does not react, brake pressure is applied automatically to provide maximum brake boost immediately once the driver does engage the brake. This functionality can usually require also short range lidar to detect obstacles in vehicle proximity (up to few meters) covering also a lateral area, as it is necessary to avoid dangerous situations for pedestrians and other road users moving with lateral relative speed. The functional architecture is shown in Figure 14.

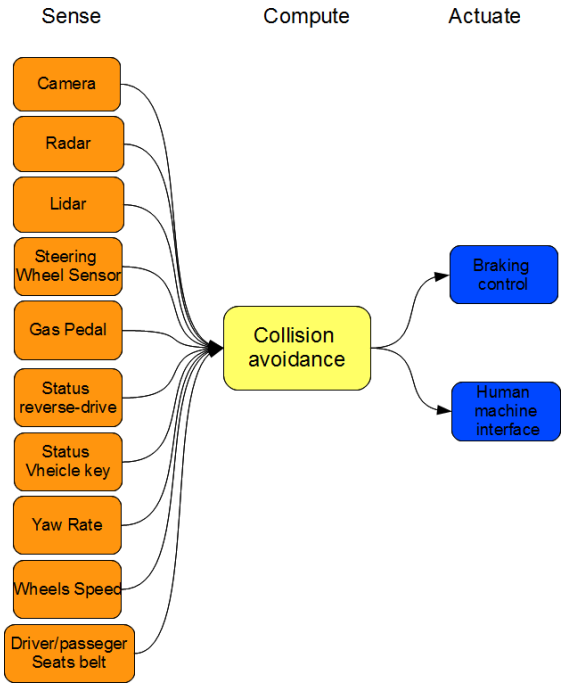

Figure 14: Functional architecture on board vehicle for collision avoidance function.

6.2.3 Vehicle dynamics control

The main shared functions to support the above functionalities are now described.

V2X Communication

Communication is a two way function to supply data and to receive information from other vehicles and infrastructures, according to the services standardized by ETSI. Communication includes:

- a firewall function, which recognizes wrong messages, and applies countermeasures against malicious attacks;
- a security check, which provides an additional barrier based on plausibility checks.

Propulsion control

By wire control of the propulsion force, by means of engine torque control and gearbox management to produce the desired acceleration.

Braking control

By wire control of the braking system, to produce the desired deceleration. This functionality includes the interaction with other braking sub-functions (e.g. ABS) or with yow rate control.

Steering torque control

This functionality consists of the superimposition of a steering torque on the steering wheel, in order to implement automatic vehicle steering, allowing at the same way any action by the driver in the case of need.

Data fusion

This functionality consists of the data fusion of the sensors collected by the different functions to provide as much as possible detailed and reliable information. Fusing multiple information sources together also produces a more efficient representation of the external environment.

Function output arbiter

This functionality defines the priority request coming from different functions to drive the specific actuation.

LoS management supervisor

This functionality defines the LoS in respect of the availability of the information in order to set the functions and their performance level according to safety rules and ensure safe vehicle operation.

6.2.4 General functional architecture

The particular functional architectures presented above can be merged into a general functional architecture, which is presented in Figure 15.

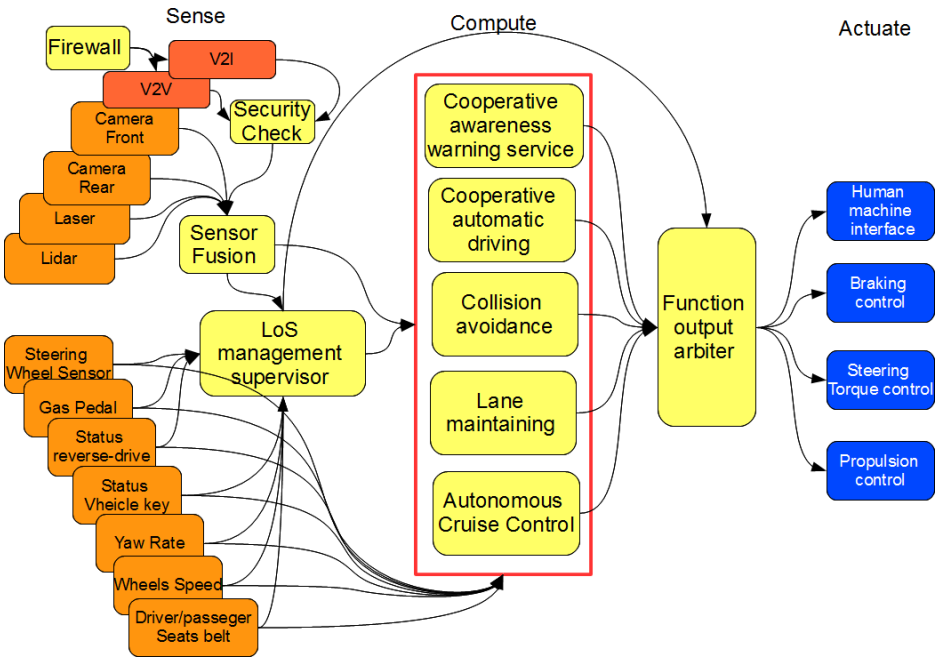

Figure 15: General functional architecture for the automotive functionalities.

6.3 Level of Service

Starting from the above defined functions, it is possible to classify them into two main categories, independently from the performance and service level, which are related to the availability of the needed information and on the operating conditions. The two categories are based on the resulting action, i.e. vehicle actuation or driver information:

- Automatic driving service
- Warning service

The level of service is a useful concept to identify different operation modes, according to the following schema.

Information source	Automatic driving services	Warning services	Level of service
V2V, V2I and onboard sensors	Co-operative driving	Cooperative awareness services	High
Onboard sensors	ADAS (vehicle control functions)	ADAS (driver information functions)	Low
Driver	Manual driving	Conventional signals	Zero

Table 3: Level of service for different operation modes.

The following principles should be pointed out:

- any level of service should be a safe state, and corresponds to the operation mode that is safe in the present operational conditions, including the available information and their quality;
- in the case of any change of the operational condition, the new safe state (or level of service) shall be reached, in a way compatible with safety criteria;
- the confidence on the information from outside shall be ensured to manage the applicable level of service;
- the level of service equal to zero represents an absolutely safe state, corresponding to a completely manual driving;
- the architectural elements that manage the level of service are not immune to failure and, therefore, shall be treated according to functional safety development process rules.

Automatic driving service

The various functionalities that can be associated to the automatic driving service are to be considered an evolution from the lower level of service of collision avoidance till the cooperative driving, where the driver is only a supervisor and the vehicle is autonomous, as illustrated in Figure 16. It is possible to associate the Level of Service at the following functions:

- **LoS 3: Cooperative automatic driving**, that includes: Overtaking manoeuvre, Platooning, Roundabout, Intersection
- **LoS 2: Autonomous cruise control**
- **LoS 1: Collision avoidance**

The Level of Service **LoS 0** is the state in which the system can be positioned after a fault is detected. This LoS is defined by means of the Hazard Analysis & Risk Assessment; a preliminary state could be: *Control function turned off, leaving the engine brake, alerting the driver that the control function is unavailable.*

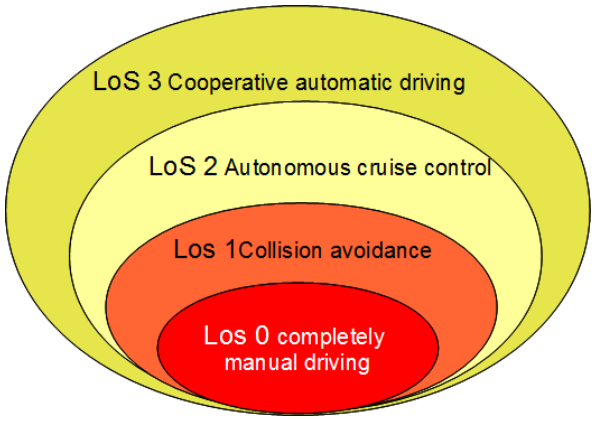

Figure 16: Evolution from LoS 0 to LoS 3 for the automatic driving service.

Warning service

Also in this case the various functionalities that can be associated to the traffic warning service are to be considered an evolution from the higher level of service of cooperative awareness till lane departure warning function. It is possible to associate the Level of Service at the following functions (see Figure 17):

- **LoS 2: The cooperative awareness**, that includes: Intersection collision warning, Signal violation warning, Lane Change Manoeuvre, Adaptive cruise control emergency

brake lights, Stationary vehicle warning, Intersection management Traffic light optimal speed advisory, Intersection management Collision Risk Warning from RSU, Intersection management Signal violation warning

- **LoS 1: Lane departure warning function**

The Level of Service **LoS 0** is the state in which the system can be positioned after a fault detected. This LoS is defined by means of the Hazard Analysis & Risk Assessment, a preliminary state could be: *Warning function turned off, alerting the driver that the warning information is unavailable*

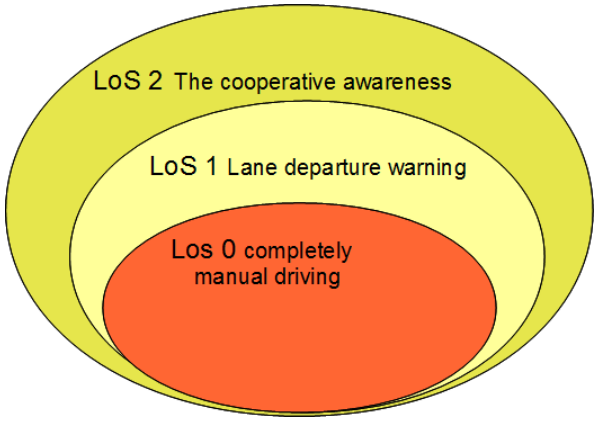

Figure 17: Evolution from LoS 0 to LoS 2 for the warning service.

6.4 Boundary of the system under safety analysis

The goal of the KARYON approach is to develop a safe system that can manage the safety critical situations caused by the unavailability of the off-board and / or on-board sensors.

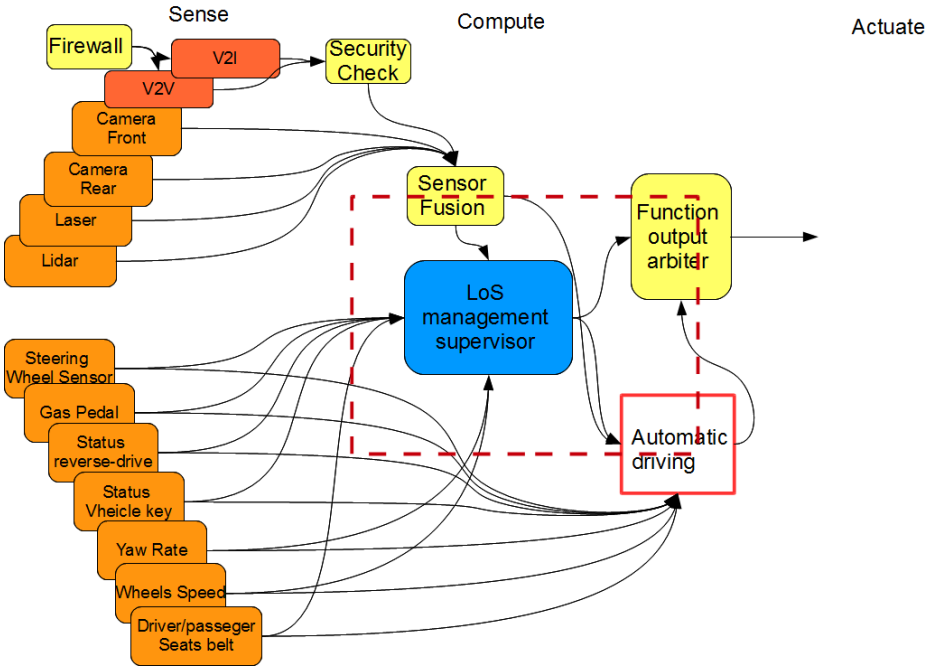

Figure 18: Boundary of the service.

To guarantee the demanded system integrity, the automatic driving system needs to be fault tolerant, reliable and safe. To set up a fault tolerant system, it is necessary to recognize faults in

an accurate way and quickly. This could be also realized by the plausibility checking system implemented in the LoS management Supervisor. Based on the information quality regarding the incoming sensor signals, it enables the controller to take adequate error handling measures, depending on the safety relevance of the system and the severity of the fault.

The structure of the modular and scalable plausibility checking system allocated into the LoS management Supervisor is shown in Figure 18.

7. Conclusions

This document is the first deliverable of WP2 and describes initial work performed in the scope of Task 2.1, Hybrid System Architecture. In particular, it provides a preliminary description of the architectural approach to manage the trade off between improved performance (and higher uncertainty) and the required safety, providing a description of the functional components in the architecture and of their interactions, which are established through data flows. The deliverable also provides a brief discussion on how the proposed architectural pattern addresses the requirements established in WP1, and discusses the most relevant implications of this architecture on other work that will need to be performed in WP3, WP4, and in Task 2.2. Finally, the deliverable also provides a preliminary application exercise, in which the architectural pattern was applied for concrete functionalities in the automotive domain. This exercise will have to be further refined and extended, and this work will allow understanding if the architectural pattern is adequately defined, or if some specific characteristics or requirements of the concrete functionalities cannot be met without changes in the generic architecture.

Given the preliminary nature of this document, our objective was essentially to provide an initial and necessarily broad overview of the main ideas and definitions that are necessary to perform other related work, namely at lower levels of abstraction. A more in-depth and complete discussion of the KARYON architectural pattern and solutions will be provided in the KARYON architecture deliverable (D2.3, to be delivered in March 2013).

References

[1] A. Casimiro, P. Martins and P. Veríssimo. *How to Build a Timely Computing Base using Real-Time Linux*. Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems (WFCS'00), pages 127–134. Porto, Portugal, September 2000.

[2] A. Casimiro, J. Kaiser and P. Veríssimo, *Generic-Event Architecture: Integrating Real-World Aspects in Event-Based Systems*, Lecture Notes in Computer Science (Architecting Dependable Systems IV), vol. 4615, pp. 287-315.

[3] M. Correia, P. Veríssimo and N. F. Neves. *The design of a COTS real-time distributed security kernel*. In Proceedings of the Fourth European Dependable Computing Conference, pages 234–252, October 2002.

[4] F. Cristian and C. Fetzer. *The Timed Asynchronous Distributed System Model*. IEEE Trans. Parallel Distributed Systems, 10(6):642–657, June 1999.

[5] C. Dwork, N. Lynch, and L. Stockmeyer. *Consensus in the presence of partial synchrony*. Journal of the ACM, 35(2):288–323, April 1988.

[6] M.J. Fischer, N.A. Lynch and M.S. Paterson. *Impossibility of Distributed Consensus with One Faulty Process*. JACM, 32(2):374–382, 1985.

[7] L. Lamport, R. Shostak and M. Pease. *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, 4(3), 382–401. 1982.

[8] L. Lamport and N. Lynch. *Distributed computing: Models and methods*. Handbook of Theoretical Computer Science, vol.B: Formal Models and Semantics. J. Van Leeuwen (Ed.), pages 1158–1199. Elsevier Science Publishers. 1990.

[9] N. F. Neves, M. Correia and P. Veríssimo. *Solving vector consensus with a wormhole*. IEEE Transactions on Parallel and Distributed Systems, 16(12):1120–1131, December 2005.

[10] H. Ortiz, A. Casimiro and P. Veríssimo. *Architecture and Implementation of an Embedded Wormhole*. In Proceedings of the 2007 Symposium on Industrial Embedded Systems (SIES'07), pages 341–344. Lisbon, Portugal, July 2007.

[11] D. Powell. *Failure mode assumptions and assumption coverage*. In Proceedings of the 22nd IEEE Annual International Symposium on Fault-Tolerant Computing (FTCS-22), pages 386–395, Boston, USA, July 1992.

[12] Trusted Computing Group, TPM Main, Part 1 Design Principles. Specification Version 1.2, Revision 62, 2003.

[13] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.

[14] P. Veríssimo and A. Casimiro. *The Timely Computing Base model and architecture*. IEEE Transactions on Computers, 51(8):916–930, 2002.

[15] P. Veríssimo. *Uncertainty and predictability: Can they be reconciled?* In Future Directions in Distributed Computing, volume 2584 of Lecture Notes in Computer Science, pages 108–113. Springer-Verlag, 2003.

[16] P. Veríssimo. *Travelling through wormholes: a new look at distributed systems models*. SIGACTN: SIGACT News (ACM Special Interest Group on Algorithms and Computation Theory), vol.37, no.1, pages 66–81, 2006.

