J Reliable Intell Environ (2015) 1:101–121
DOI 10.1007/s40860-015-0013-3

ORIGINAL ARTICLE

Antifragility: systems engineering at its best

Eric Verhulst[1] · **Bernhard Sputh**[1] · **Pieter Van Schaik**[1]

Received: 14 October 2015 / Accepted: 29 October 2015 / Published online: 17 November 2015

Abstract Systems engineering has emerged because of the growing complexity of systems and the growing need for systems to provide a reliable service. The latter has to be defined in a wider context of trustworthiness and covering aspects like safety, security, human–machine interface design and even privacy. What the user expects is an acceptable quality of service (QoS), a property that is difficult to measure as it is a qualitative one. In this paper, we present a novel criterion, called assured reliability and resilience level (ARRL) that defines QoS in a normative way, largely by taking into account how the system deals with faults. ARRL defines 7 levels of which the highest one can be described as the level where the system becomes antifragile.

1 Introduction

One of the emerging needs of embedded systems is better support for safety and, increasingly so, security. These are essentially technical properties. The underlying need is trustworthiness. This covers not only safety and security, but also aspects of privacy and usability. All of these aspects can be considered as specific cases of the level of trust that a user or stakeholder expects from the system. When these are lacking we can say that the system has failed or certainly resulted in a dissatisfied user. The effects can be catastrophic with loss of lives and costly damages, but also simply annoyance that ultimately will result in financial consequences for the producer of the system. To achieve the desired properties, systems engineering standards and in particular safety standards were developed. These standards do not cover the full spectrum of trustworthiness. They aim to guarantee safety properties because they concern the risk that people are hurt or killed and the latter is considered a higher priority objective than all other ones (at least today). It is because of said risk that safety-critical systems are generally subjected to certification as a legal requirement before putting them in public use. In this paper, we focus on the safety engineering aspects, but the analysis can be carried over to the other domains and critical properties as well.

While safety standards exist, a first question that arises is why each domain has specific safety standards [1]. They all aim to reduce the same risk of material damages and human fatalities to a minimum, so why are they different from one domain to another? One can certainly find historical reasons, but also psychological ones. Safety standards are often associated with or concern mostly systems with programmable electronic components. For example, IEC 61508 [2]—the so-called mother of all safety standards—explicitly addresses systems with programmable components. The reason for this is that with the advent of programmable components in system design, systems engineering became dominantly a discrete domain problem, whereas the preceding technologies were dominantly in the continuous domain. In the continuous domain components have the inherent property of graceful degradation, while this is not the case for discrete domain systems. A second specific trait is that, in the discrete domain, the state space is usually very large with state changes that can happen in nanoseconds. Hence it is

✉ Eric Verhulst
eric.verhulst@altreonic.com

Bernhard Sputh
bernhard.sputh@altreonic.com

Pieter Van Schaik
pieter.vanschaik@altreonic.com

[1] Altreonic NV, Gemeentestraat 61A bus 1, 3210 Linden, Belgium

very important to be sure that no state change can bring the system into an unsafe condition. Despite identifiable weaknesses, different from domain to domain, safety engineering standards impose a controlled engineering process resulting in fairly well predictable safety that can be certified by external parties. However, the process is relatively expensive and essentially requires that the whole project and system is re-certified whenever a change is made. Similarly, a component such as a general purpose computer that is certified as safe to use in one domain cannot be reused as such in another domain. The latter statement is even generous. When strictly following the standards, within the same domain each new system requires a re-certification or at least a re-qualification, so that even within product families reuse is limited by safety concerns.

Many research projects have already attempted to address the issues, whereby a first step is often trying to understand the problem. Two projects were inspirational in this context. A first project was the ASIL project [3]. It analyzed multiple standards like IEC-61508, IEC-62061, ISO-26262, ISO-13849, ISO-25119 and ISO-15998 as well as CMMI and Automotive SPICE with the goal to develop a single process flow for safety-critical applications, focusing on the automotive and machinery sectors. This was mainly achieved by dissecting the standards in a semi-atomic way whereby the paragraphs were tagged with links to an incrementally developed V-model of the ASIL flow. In total this has resulted in more then 3000 identified process requirements and about 100 different work products (so-called artefacts required for certification). The process itself contains about 350 steps divided in organizational processes, development and supporting processes. The project demonstrated that a unifying process flow compatible with multiple safety standards is achievable although tailoring is not trivial. The ASIL flow was also imported in the GoedelWorks portal [4]. The latter is based on a generic systems engineering metamodel, demonstrating that using a higher-level abstract model for system engineering (in casu, safety engineering) is possible. At the same time it made a number of implicit assumptions explicit. For example, the inconsistent use of terminology and concepts across different domains is a serious obstacle to reuse.

A second project that is now terminated was the FP7 OPENCOSS project [5]. It aimed at reducing the cross-domain and cross-product certification or safety assessment costs. In this case the domains considered are avionics, railway and automotive. The initial results have amongst other shown how vast the differences are in applying safety standards into practical processes. The different sectors are also clearly at different levels of maturity in adopting the safety standards, even if generally speaking the process flows are similar. The project focused not so much on analyzing the differences but on coming up with a common metamodel (the so-called CCL or Common Certification Language) that supports building up and retrieving arguments and evidence from a given project with the aim to reuse these for other safety-critical projects. The argument pattern used is provided by the GSN [6] notation.

Hence, both projects have provided the insight that strictly speaking cross-domain reuse of safety-related artefacts and components is not possible due to the vast differences between the safety standards. In what follows we will see that the notions of safety as a goal (often called the safety integrity level or SIL) are different from one domain to another. This can be justified. The safety assurance provided for a given system is specific to that system in its certified configuration and its certified application. This is often in contrast with the engineering practice. Engineers constantly build systems by reusing existing components and composing them into larger subsystems. This is not only driven by economic benefits, but it often increases the trust in a system because the risk for residual errors will be lower, at least if a qualification process for these components is in use. Nevertheless, engineering and safety standards contain relatively very few rules and guidelines on reusing components hampering the development of safe systems by composition.

Another system aspect that is emerging is lifecycle management. This aspect comes to the foreground as systems have increasingly longer lifetimes, whereby the components are more and more connected in a larger system. In addition, the lifetimes are such that individual components will be replaced or upgraded as they age or can be replaced with more novel, more efficient or more performant technologies. Typical examples are Internet Of Things, smart grids and infrastructure that becomes "smart" by using embedded electronics. The distinguishing feature is that a developed system can no longer be developed in isolation as it becomes part of a system of systems.

This paper analyzes why the current safety-driven approach is unsatisfactory for reaching trustworthiness as a lifecycle goal for systems. It introduces a new criterion called the assured reliability and resilience level (ARRL) that allows components to be reused in a normative way while preserving the safety integrity levels at the system level. The higher ARRL level break out of the component domain and define the system itself as a component in a larger system that includes it operating environment. ARRL-6 and -7 include the definition of a process that must be in place and aims at continuously improving the system. This property corresponds to the notion of antifragility as originally formulated as a qualitative concept by Taleb [7].

2 Safety integrity levels

As safety is a critical property, it is no wonder that safety standards are perhaps the best examples of concrete systems engineering standards, even if safety is not the only property that is relevant for systems engineering projects. Most domains have their own safety standards partly for historical reasons, partly because the heuristic knowledge is very important or because the practice in the domain has become normative. We consider first the IEC 61508 standard, as this standard is relatively generic. It considers mainly programmable electronic systems (Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES). The standard consists of 7 parts and prescribes 3-stage processes divided in 16 phases. The goal is to bring the risks to an acceptable level by applying safety functions. IEC 61508 starts from the principle that safety is never absolute; hence it considers the likelihood of a hazard (a situation posing a safety risk) and the severity of the consequences. A third element is the controllability. The combination of these three factors is used to determine a required SIL or Safety Integrity Level, categorized in 4 levels, SIL-1 being the lowest and SIL-4 being the highest. These levels correspond with normative allowed Probabilities of Failure per Hour and require corresponding Risk Reduction Factors that depend on the usage pattern (infrequent versus continuous). The risk reduction itself is achieved by a combination of reliability measures (higher quality), functional measures as well as assurance from following a more rigorous engineering process. The safety risks are in general classified in 4 classes, roughly each corresponding with a required SIL level whereby we added a SIL-0 for completeness. This classification can easily be extended to economic or financial risks. Note, that we use the term SIL as used in IEC 61508, while the table is meant to be domain independent (Table 1).

Table 1 Categorization of safety risks

Category	Typical SIL	Consequence upon failure
Catastrophic	4	Loss of multiple lives
Critical	3	Loss of a single life
Marginal	2	Major injuries to one or more persons
Negligible	1	Minor injuries at worst or material damage only
No consequence	0	No damages, except user dissatisfaction

The SIL level is used as a directive to guide selecting the required architectural support and development process requirements. For example, SIL-4 imposes redundancy and positions the use of formal methods as highly recommended.

While 61508 has resulted in derived domain-specific standards (e.g. ISO 26262 for automotive [8], EN 50128 [9] for railway), there is no one to one mapping of the domain-specific levels to IEC-61508 SIL levels. Table 2 shows an approximate mapping, whereby we added the aviation DO-178C [10] standard that was developed from within the aviation domain itself. It must be mentioned that the Risk Reduction Factors are vastly different as well. This is mainly justified by the usage pattern of the systems and the accepted "fail safe" mode. For example, while a train can be stopped if a failure is detected, a plane must at all cost be kept in the air in a state that allows it still to land safely. Hence, Table 2 is not an exact mapping of the SIL levels but an approximate one. However, in general each corresponding level will require similar functional safety support, similar architectural support as well as higher degrees of rigor in the development process followed, even if the risk reduction factors are quantitatively different.

The SIL levels (or the domain-specific ones) are mostly determined during a HARA (hazard and risk analysis) executed before the development phase and updated during and after the development phase. The HARA tries to find all Hazardous situations and classifies them according to 3 main criteria: probability of occurrence, severity and controllability. This process is however difficult and complex, partly because the state space explodes very quickly, but also because the classification is often not based on historical data (absent for any new type of system) but on expert opinions. It is therefore questionable if the assigned safety levels are accurate enough and if the risk reduction factors are realistic, certainly for new type of systems. We elaborate on this further.

Once an initial architecture has been defined, another important activity is executing an FMEA (failure mode effect

Table 2 Approximate cross-domain mapping of safety integrity levels

Domain	Domain-specific safety levels				
General (IEC-61508) Programmable electronics	(SIL-0)	SIL-1	SIL-2	SIL-3	SIL-4
Automotive (26262)	ASIL-A	ASIL-B	ASIL-C	ASIL-D	–
Aviation (DO-178/254)	DAL-E	DAL-D	DAL-C	DAL-B	DAL-A
Railway (CENELEC 50126/128/129)	(SIL-0)	SIL-1	SIL-2	SIL-3	SIL-4

analysis). While a HARA is top-down and includes environmental and operator states, the FMEA analyzes the effects of a failing component on the correct functioning of the system (and in particular in terms of the potential hazards). Failures can be categorized according to their origin. Random failures are typically a result of physical causes, whereas systematic failures are a result of either design or implementation errors. In all cases, when programmable electronics are used, their effect is often the same: the system can go immediately or in a later time interval into an unsafe state. It is also possible that single or even multiple faults have accumulated but remain latent until the error is triggered by a specific event.

In all cases only an adequate architecture can intercept the failures before they generate further errors and hence pose a safety risk. As such the HARA and FMEA will both define safety measures (like making sure that sensor data correspond to the real data even when a sensor is defective). While the HARA, being executed prior to defining an architecture, should define the safety measures independently of the chosen implementation architecture, the FMEA will be architecture dependent and hence also related to the components in use. The results of the FMEA are not meant to be reused in a different system, even if the analysis is likely generic enough to support the reuse in other systems. As such, there is no criterion defined that allows us to classify components in terms of their trustworthiness, even if one can estimate some parameters like MTBF (mean time between failures) albeit in a given context. In the last part of this paper, we introduce a criterion that takes the fault behavior into account. Note that while generic, it should be clear that the focus of this paper is on software components running on programmable electronic components. This will be justified further.

2.1 Quality of service levels

An inherent weakness from the systems engineering and user's point of view is that trustworthiness, in all its aspects, is not the only property of a system. A system that is being developed is part of a larger system that includes the user (or operator) as well as the environment in which the system is used, see Fig. 1.

A HARA, for example, looks primarily at the safety risks that can originate in any of these three contextual systems. Both additional systems do not necessarily interact in a predictable way with the envisioned system and have an impact on the safety properties and assurance. Note that we can also consider security risks as a subtype of safety risk, the difference being the origin of the resulting fault (maliciously injected versus originating in the system or its operating environment).

From the user's point of view, the system must deliver an acceptable and predictable level of service, which we call the quality of service (QoS). A failure in a system is not seen as an immediate safety risk but rather as a breach of contract on the QoS, whereby the system's malfunction can then result in a safety-related hazard or a loss of mission control, even when no safety risks are present. As such we can see that a given SIL is a subset of the QoS. The QoS can be seen as the availability of the system as a resource that allows the user's expectations to be met. Aiming to reduce the intrinsic ambiguities of the safety levels we now formulate a scale of QoS as follows:

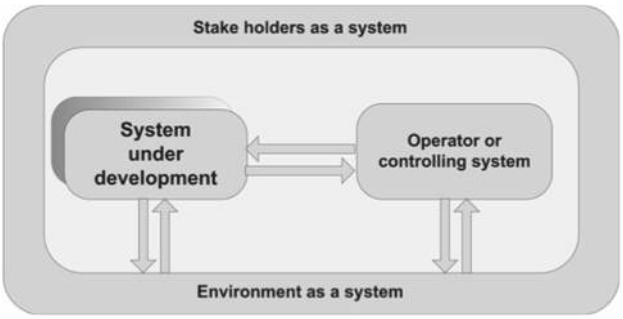

Fig. 1 The context in which a system under development is used

- QoS-1 is the level whereby there is no guarantee that there will always will be enough resources to sustain the service. Hence the user should not rely on the system and should consider it as untrustworthy. When using the system, the user is taking a risk that is not predictable.
- QoS-2 is the level whereby the system must assure the availability of the resources in a statistically acceptable way. Hence, the user can trust the system but knows that the QoS will be lower from time to time. The user's risk is mostly one of annoyance and dissatisfaction or of reduced service.
- QoS-3 is the level whereby the system can be trusted to always have enough resources to deliver the highest QoS at all times. The user's risk is considered to be negligible.

We can consider this classification to be less rigorous than the SIL levels, because it is based on the user's perception of trustworthiness and not on a combination of probabilities even when these are questionable (see Sect. 4). On the other hand, QoS levels are more ambitious because they define minimum levels that must be assured in each QoS level. Of course, the classification leaves room for residual risks; but those are not considered during the system's design but are accepted as uncontrollable risks with negligible probabilities. Neither the user nor the system designer has much control over them. Perfect systems that never encounter issues are physically not possible.

2.2 Some data for thought

While risks associated with health and political conflicts are still very dominant as cause of death and injuries, technical risks like working in a factory or using a transportation system are considered more important because they have a higher emotional and visible economic cost, even if the number of fatalities is statistically low. The reason is probably because the perception is that these risks are avoidable and hence a responsible party can be identified, eventually resulting in financial liabilities.

As a result, sectors like railway and aviation are statistically very safe. As an example, about 1000 people are killed every year worldwide in aircraft-related accidents, which makes aviation the safest transport mode in the world [11]. In contrast the automotive sector adds up to about 1.2 million fatalities per year worldwide and even developed regions like the USA and Europe experience about 35,000 fatalities per year (figures for 2010) [12]. These figures are approximate, as the statistics certainly do not include all causalities. Although both sectors have their safety standards, there is a crucial difference. Whereas in most countries aircrafts and railway systems are strictly regulated and require certification, in the automotive sector the legal norms are much weaker partly because the driver is considered as the main cause of accidents. The latter biases significantly the "controllability" factor in the required SIL determination.

Taking a closer look at the SIL classifications of IEC 61508 and the automotive derived ones in ISO-26262, we notice three significant differences:

1. Whereas IEC-61508 and ISO-26262 both define 4 levels, they do not map to each other—in particular SIL-3 and SIL-4 do not map to ASIL-C and -D.
2. The highest ASIL-D level corresponds to a SIL-3 level in terms of casualties, although it is not clear if this means a few casualties (e.g. not more than five like in a car) or several hundreds (like in an airplane.)
3. The aviation industry experiences about 1000 casualties per year worldwide, whereas the automotive industry experiences 1200 times more per year worldwide, yet the ASIL level are de facto lower.

When we try to explain these differences, we can point to the following factors:

1. ISO-2626 was defined for automotive systems that have a single central engine (at least that is still the prevailing vehicle architecture). As a direct consequence of the centralized and non-redundant organization such a vehicle cannot be designed to be fault-tolerant (which would require redundancy) and therefore cannot comply with SIL-4 (which mandates a fault-tolerant design).
2. While ASIL-C more or less maps onto SIL-3 (upon a fault the system should transition to a fail-safe state), ISO-26262 introduces ASIL-C requiring a supervising architecture. In combination with a degraded mode of operation (e.g. limp mode), this weaker form of redundancy can be considered as fault tolerant if no common mode failure affects both processing units [13].
3. Automotive systems are not (yet) subjected to the same stringent certification requirements as railway and aviation systems, whereby the manufacturers as well as the operating organization are legally liable, whereas in general the individual driver is often considered the responsible actor in case of an accident. Note that, when vehicles are used in a regulated working environment, the safety requirements are also more stringent, whereby the exploiting organization is potentially liable and not necessarily the operator or driver. Hence, this lesser financial impact of consumer-grade products is certainly a negative factor even if the public cost price is high as well.
4. The railway and aviation sectors are certified in conjunction with a regulated environment and infrastructure that contributes to the overall safety. Automotive vehicles are engineered with very little requirements in term of where and when they are operated and are used on a road infrastructure that is developed by external third parties. The infrastructure which cars share forces them to be operated in a much closer physical proximity than airplanes. This in turn also increases the probability of accidents. This partly explains why the high number of worldwide casualties is not reflected in the ASIL designation.
5. One should not conclude from the above that a vehicle is hence by definition unsafe. Many accidents can be attributed to irresponsible driving behavior. It is however a bit of a contradiction that the Safety Integrity Levels for automotive are lower than those for aviation and railway if one also considers the fact that vehicle accidents happen in a very short time interval and confined spaces with almost no controllability by the driver. In railway and aviation often minutes and much space are available for the driver or pilot to attempt to regain control.
6. ISO-26262 also defines guidelines for decomposing a given ASIL level. However, the process is complex and is driven by an underlying goal to reduce the cost supported by the rationale that simultaneous failures are not likely. The latter assumption is questionable.

2.3 The weaknesses in the application of the safety integrity levels

As we have seen above, the use of the safety integrity levels does not result in univocal safety. We can identify several weaknesses:

1. A SIL level is a system property derived from a prescribed process whereas systems engineering is a mixture of planning, prescribed processes and architecting/developing. As such a SIL level is not a normative property as it is unique for each system.
2. SIL levels are the result of probabilities and estimations, while analytical historical data are not always present to justify the numbers. Also here we see a difference between the automotive domain and the aviation and railway domains. The latter require official reporting of any accident; have periodic and continuous maintenance schedules (even during operational use) and the accidents are extensively analyzed and made available to the community. Black boxes are a requirement to allow post-mortem analysis. Nevertheless, when new technologies are introduced the process can fail, as was recently demonstrated by the use of Lithium-ion batteries and Teflon cabling by Boeing [14].
3. SIL levels, defined as a system level property, offer little guidance for reusing and selecting components and sub-system modules whereas engineering is inherently a process whereby components are reused. An exception is the ISO-13849 machinery standard and its derivatives, all IEC-61508 derived standards. Also the aviation sector has developed a specific standardized IMA architecture described in the DO-197 standard that fosters reuse of modular avionics, mainly for the electronic on board processing [15]. ISO-26262 also introduced the notion of a reusable component called SEooC (safety element out of context) allowing a kind of pre-qualification of components when used in a well-specified context. While we see emerging notions of reuse in the standards, in general very little guidance is offered on how to achieve a given SIL level by composing systems from different components. The concept is there, but not yet formalized.
4. An increasing part of safety-critical systems contain software. Software as such has no reliability measures, only residual errors while its size and non-linear complexity is growing very rapidly, despite efforts in partitioning and layering approaches that rather hide than address the real complexity. This growth is not matched by an equal increase in controllability or productivity [16]. If one of the erroneous (but unknown) states is reached (due to a real program error or due to an external hardware disturbance) this can result in a safety risk. Such transitions to an erroneous state cannot be estimated up front during a SIL determination. In addition, new advanced digital electronics and their interconnecting contacts do not have well known reliability figures. They are certainly subject to aging and stress (like analog and mechanical components), but they can fail catastrophically in a single clock pulse measured in nanoseconds.
5. The SIL level has to be seen as the top-level safety requirement of a system. In each application domain different probabilistic goals (in terms of risk reduction) are applied with an additional distinction between intermittent and continuous operation. Hence cross-domain reuse or certification can be very difficult, because the top level SIL requirements are different, even if part of the certification activities can be reused.
6. A major weakness of the SIL is however that it is based on average statistical values, with often no information on the statistical spread. Not only are correct figures very hard or even impossible to obtain, they also depend on several factors such as usage pattern, the operating environment, and the skills and training of the human operator. Correct statistical values such as the mean value assume a large enough sampling base, which is often not present. Moreover it ignores that disruptive events like a very unlikely accident can totally change these values. As an example we cite the Concorde airplane that was deemed to be the safest aircraft in the world until one fatally crashed. After the catastrophic event it "became" almost instantly one of the most unsafe airplanes in the world, at least statistically speaking, partly because the plane was less intensively used than most commercial planes.

The last observation is crucial. While statistical values and estimations are very useful and essential design parameters, very low residual risks can still have a very high probability of happening. We call this the Law of Murphy: if something can happen, eventually it will happen. Referring to their low statistical probability will not save lives. The estimated probability can be very different from the one observed after the facts. The latter

2.4 SIL calculations and non-linearity

SIL determination in the standards is often based on statistical values such as the probability of occurrence and semi-subjective estimations of the severity of the hazard and the controllability. While a human operator can often be a crucial element in avoiding a catastrophe, it is also a subjective and uncontrolled factor; hence it should be used with caution as an argument to justify lesser functional risk reduction efforts. In addition there is a gray zone whereby the human operator might be seen as having inadequately reacted, but a deeper analysis will often highlight ambiguities and con-

Table 3 Technology levels in a system

Technology level	Dominant property	Dominant fault types	Typical safety measures
Environment	External constraints	Unforeseen interactions	Co-design of infrastructure and system
Operator/user	Human interaction	Human–machine interface confusion	Analysis of HMI and testing. Adequate training
Software	Discrete state-space, non-linear time	Design faults and logical errors	Redundancy and diversity at macro-level, formal correctness
Electronics	Combinatorial state-space, discrete time	Transient faults and wear out	Redundancy at micro-level
Material	Mainly continuous or linear properties	Permanent or systemic faults	Adding robustness safety margin

fusion generated by the user interface subsystem [17]. In general, in any system with software programmable components, we can distinguish three levels of technology, as well as 2 external domains as summarized in Table 3. In terms of safety engineering, one must also take into account the human operator and the environment in which the system is used. They mainly impose usage constraints on the safe use of the system.

We can now see more clearly why safety standards think mostly in terms of probabilities and quality. In the days before programmable electronics, system components were "linear", governed by material properties. One only has to apply a large enough safety margin (assuming an adequate architecture) whereby an observable graceful degradation acts as a monitoring function. Non-linearities (i.e., discontinuities) can happen if there is a material defect or too much stress. Electronic devices are essentially also material devices and are designed with the same principles of robustness margins, embedded in technology-specific design rules.

With the introduction of digital logic, a combinatorial state machine was introduced and a single external event (e.g. a charged particle) could induce faults. The remedy is redundancy at the micro-level: parity bits, CRC codes, etc. Note however that digital logic is not so linear anymore. It goes through the state machine in steps and a single faulty bit can lead to an erroneous illegal state or numerical errors.

Software makes this situation worse as now we have an exponentially growing state machine. In addition software is a non-linear system. Every clock pulse the state is changed and even the execution thread can switch to another one. The remedy is formal proof (to avoid reaching undesired states) and redundancy (but with diversity).

Each of the levels actually depends on the lower levels, whereby we have the special situation that software assumes that the underlying hardware is perfect and fault free. Any error in software is either a design or an implementation error, whereby the cause is often an incomplete or ambiguous specification, or a hardware induced fault. Therefore, reasoning in terms of probabilities and quality degrees for digital electronics and software has value but means little when using it as a safety-related design parameter. In the discrete domain a component is either correct or not correct, whereby we use the term correct in the sense of being free of errors. While we can reduce the probability of reaching an erroneous illegal state by means of, for instance, using a better development process or a better architecture, the next event (external or internal state transition like the on-chip clock) can result in a catastrophic outcome. This must be the starting point for developing safe systems with discrete components if one is really serious about safety. In general, graceful degradation does not apply to discrete state space systems.

3 The missing link in safety engineering: the ARRL criterion

Despite the weaknesses of the SIL criterion, safety standards are still amongst the best of the available engineering standards and practices in use. In addition, those standards contain many hints on how to address safety risks, though not always in an outspoken way.

As an example, every standard outlines safety preconditions. The first one is the presence of a safety culture. Another essential principle in safety engineering is to avoid any unnecessary complexity. In formal terms: keeping the project's and system's state space under control. A further principle is that quality and the resulting reliability come before safety otherwise any safety measure becomes unpredictable. This is reflected in the requirements for traceability and configuration management. We focus on the last one to define a novel criterion for achieving safety by composition. Traceability and configuration management are only really possible if the system is developed using principles of orthogonal composability, hence we need modular architectures whereby components are (re)-used that carry a trustworthiness label. Trustworthiness is here meant to indicate that the component meets its specifications towards the external interface it presents to other components. We can call this the

Table 4 ARRL levels

ARRL level	ARRL definition
ARRL-0	The component might work ("use as is"), but there is no assurance. Hence all risks are with the user
ARRL-1	The component works as tested, but no assurance is provided for the absence of any remaining issues
ARRL-2	The component meets all its specifications, if no fault occurs. This means that it is guaranteed that the component has no implementation errors, which requires formal evidence as testing can only uncover testable cases. The component still provides ARRL-1 level assurance by testing as also formal evidence does not necessarily provide complete coverage but should uncover all so-called systematic faults, e.g., a wrong parameter value. In addition, the component can still fail due to randomly induced faults, for example an externally induced bit-flip
ARRL-3	The component inherits all properties of the ARRL-2 level and in addition is guaranteed to reach a fail-safe or reduced operational mode upon a fault. This requires monitoring support and some form of architectural redundancy. Formally speaking this means that the fault behavior is predictable as well as the subsequent state after a fault occurs. This implies that specifications include all fault cases as well as how the component should deal with them
ARRL-4	The component inherits all properties of the ARRL-3 level and can tolerate one major fault. This corresponds to requiring a fault-tolerant design. This entails that the fault behavior is predictable and transparent to the external world. Transient faults are masked out
ARRL-5	The component inherits all properties of the ARRL-4 level but is using heterogeneous sub-components to handle residual common mode failures
Inheritance rule	The component inherits all properties of any lower level ARRL properties

component's contract it presents to its ambient environment. In addition, in practice many components are developed independently of the future application domain (with the exception of for instance normative parameters for the environmental conditions). The conclusion is clear: we need to start at the component level and define a criterion that gives us a normative definition. We also need reusability guidance on how to develop components in a way that allows reusing them with no negative impact on safety at the system level.

In previous sections we have shown why SIL might not be a suitable criterion. In the attempt to deal with the shortcomings of SIL in what follows we introduce the ARRL or Assured Reliability and Resilience Level to guide us in composing safe systems. The different ARRL classes are defined in Table 4. They are mainly differentiated in terms of how much assurance they provide in meeting their contract in the presence of faults.

Before we elaborate on the benefits and drawbacks of the ARRL criterion, we should mention that there is an implicit assumption about a system's architecture. A system is composed by defining a set of interacting components. This has important consequences:

1. The component must be designed to prevent the propagation of errors. Therefore the interfaces must be clearly identifiable and designed with a "guard". These interfaces must also be the only way a component can interact with other components. The internal state is not accessible from another component, but can only be made available through a well-defined protocol (e.g. whereby a copy of the state is communicated).
2. The interaction mechanism, for example a network connection, must carry at least the same ARRL credentials as the components it interconnects. Actually, in many cases, the ARLL level must be higher if one needs to maintain a sufficiently high ARRL level at the level of the (sub)-system composed of the components.
3. Hence, it is better to consider the interface as a component on itself, rather than for example assuming an implicit communication between the components.

Note that when a component and its connected interfaces meet the required ARRL level, this is a required precondition, not a sufficient precondition for the system to meet a given ARRL and SIL level. The application itself developed on top of the assembled components and its interfaces must also be developed to meet the corresponding ARRL level.

4 Discussion of the ARRL levels

By formalizing the ARRL levels, we make a few essential properties explicit:

- Inheritance: high level ARRLs inherit the properties of a lower level ARRL. This is essential for the resilience properties.
- The component must carry evidence that it meets its specifications. Hence the use of the "Assured" qualifier. Without evidence, no verifiable assurance is possible. The set of assured specifications, that includes the assumptions and boundary conditions, can be called the contract

fulfilled by the component. In addition, verifiable and supporting evidence must be available to support the contract's claims.
- Reliability is used to indicate the need for a sufficient quality of the component. A high reliability implies that the MTBF will be high (in terms of its lifetime) and is hence not a major issue in using the component.
- Resilience is used to indicate the capability of the component to continue to provide its intended functionality in the presence of faults. This implies that fault conditions can be detected, their effects mitigated and error propagation is prevented [18].
- There is no mentioning of safety or security levels because these are system level properties that also include the application specific functionality.
- The ARRL criterion can be applied in a normative way, independently of the application domain. The contract and its evidence for it should not include domain-specific assumptions.
- By this formalization we also notice that the majority of the components (software or electronic ones) on the market will only meet ARRL-1 (when tested and a test report is produced). ARRL-2 assumes the use of formal evidence and very few software products meet these requirements. From ARRL-3 on, a software component has to include additional functionality that deals with error detection and isolation and requires a software-hardware co-design. With ARRL-4 the system's architecture is enhanced by explicitly adding redundancy and whereby it is assumed that the faults are independent in each redundant channel. In software, this corresponds to the adoption of design redundancy mechanisms so as to reduce the chance of correlated failures.
- When a component has a fault its ARRL level drops into a degraded mode with a lower ARRL level. For the higher ARRL levels this means that the functionality can be preserved but its assurance level will drop. This is achieved by making the fault behavior explicit and hence verifiable.
- The SIL levels as such are not affected.

ARRL-5 further requires 3 quasi-independent software developments on different hardware, because ARRL-4 only covers a subset of the common mode failures. Less visible aspects are for instance common misunderstanding of requirements, translation tool errors and time dependent faults. The latter require asynchronous operation of the components and diversity using a heterogeneous architecture.

5 ARRL architectures illustrated

While Table 3 discusses several technology levels in a system or component, the focus is on the hardware (electronics) and software levels. The lowest level is largely the continuous domain where the rules and laws of material science apply. In general, this domain is well understood and applying design and safety margins mitigates most safety risks. In addition, components in this domain often exhibit graceful degradation, a property that inherently contributes to safety. This even applies to the semiconductor materials used for developing programmable chips.

The levels related to the environment and the user/operator of a system are mostly related to external factors that can create hazardous situations. Hence these must be considered when developing the system and they play an important role in the HARA. However, as such these are external and often unique factors for every system, the reuse factor (except for example in identifying reusable patterns and scenarios) is limited.

In this paper, the focus is on how a component or subsystem can be reused in the context of a safety-critical application. This is mostly an issue in the hardware and software domains because these technology domains are characterized by very large state spaces. In addition, as mentioned before, such systems often will operate in a dynamic and reconfigurable way. In addition, a component developed in these discrete technologies can fail practically speaking in a single instant in time. To mitigate these risks, ARRL levels explicitly take the fault behavior into account as well as the desired state after a fault occurred. This results in derived requirements for the architecture of the component, the contract it carries as well as for the evidence that supports it. Therefore the evidence will also be related to the process followed to develop the component. To clarify the ARRL levels, a more visual representation is used and discussed below.

5.1 The ARRL component view

Figure 2 illustrates the generic view of a component. It is seen as a functional block that accepts input vectors, processes them and generates output vectors. In the general sense, the processing can be seen as the transfer function of the component. While the latter terminology is mostly used in the continuous domain, in the discrete domain the transfer function is often a state machine or a collection of concurrent state machines. Important for the ARRL view is that the processing function is not directly linked with the inputs and outputs but via component interfaces that operate as guards.

5.2 An illustrated ARRL-1 component

As the ARRL-0 provides no assurance at all for its behavior, we can gracefully skip this level, hence we start with the ARRL-1 level (Fig. 3). Such a component can only be partially "trusted", i.e. as far as it was tested. The uncer-

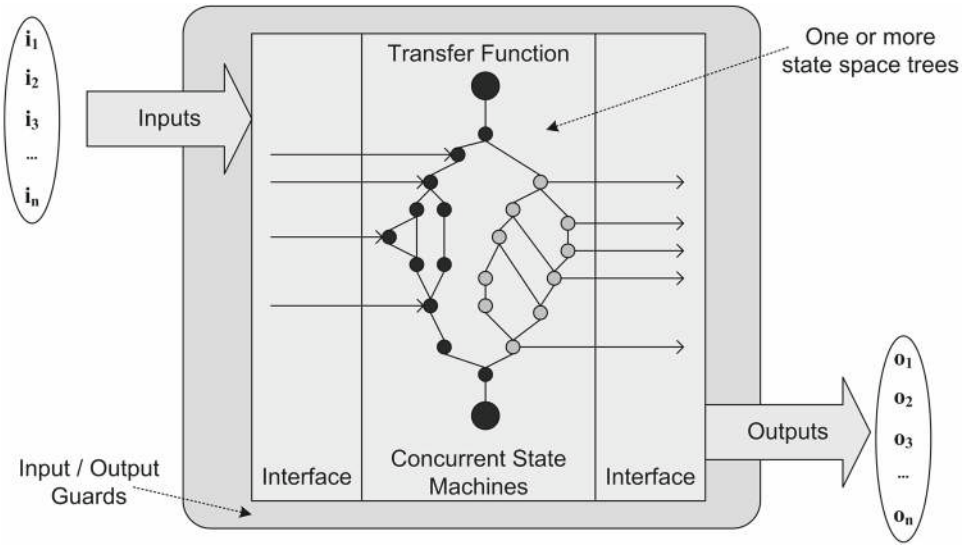

Fig. 2 ARRL generic view of a component

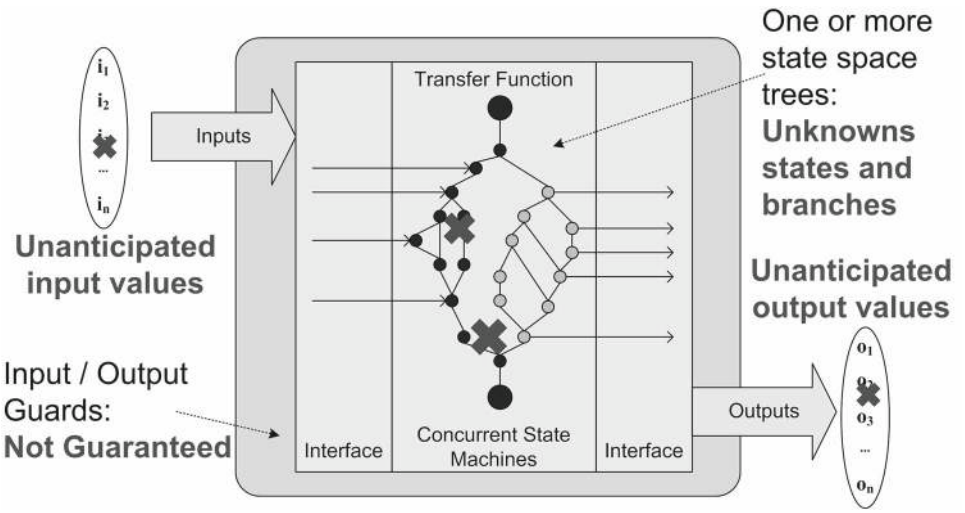

Fig. 3 A generic ARRL-1 component

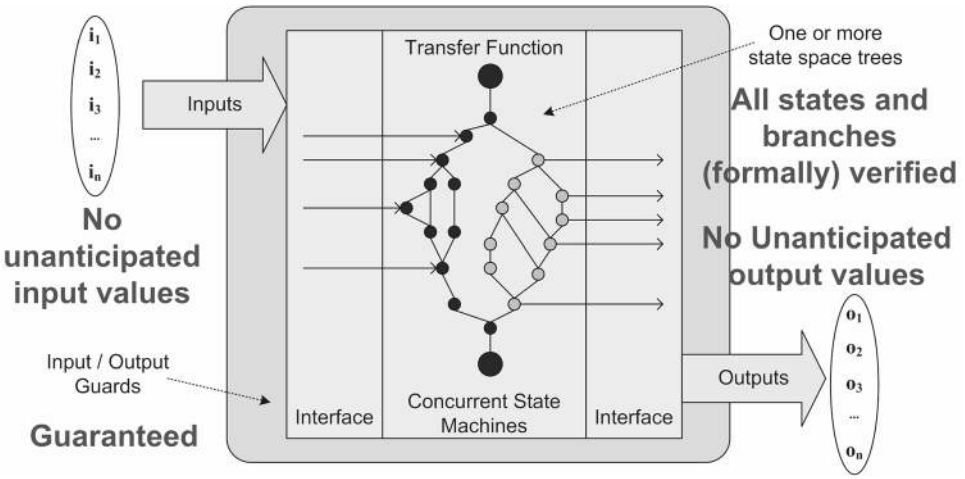

Fig. 4 A generic ARRL-2 component

tainty is related to unanticipated input values; doubts that the input/output guards are complete, remaining errors in the processing function, invalid assumptions (e.g. erroneous requirements [19]) and hence there can be unanticipated output values. In other words, while a test report provide some evidence, the absence of errors if not guaranteed and as such a

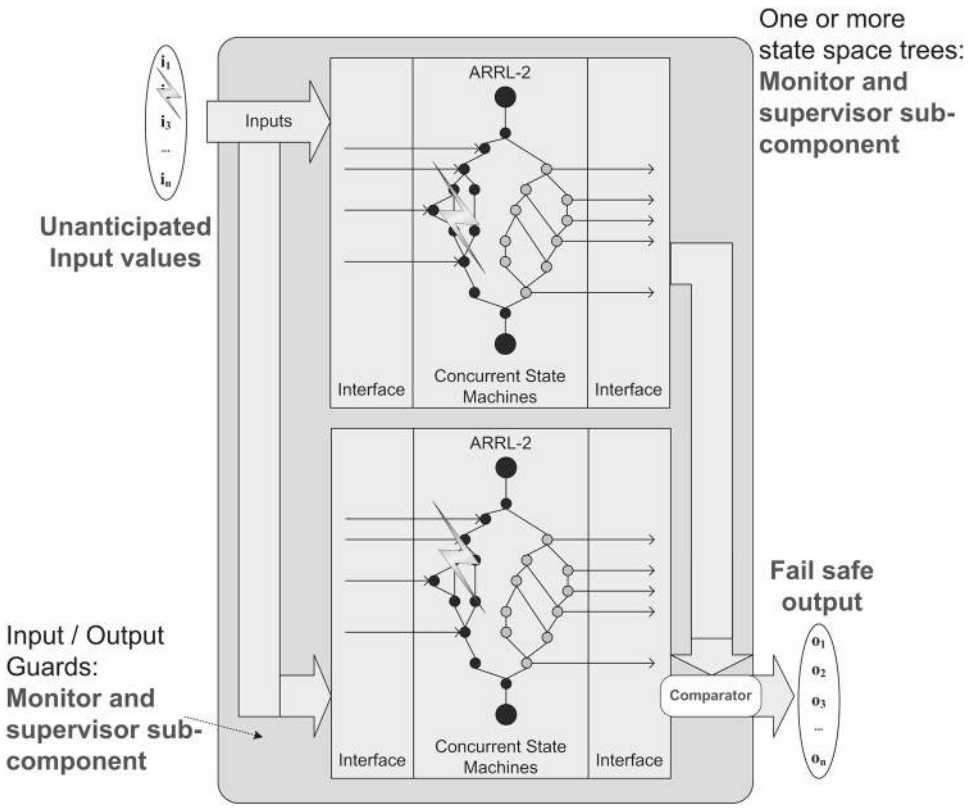

Fig. 5 A generic ARRL-3 component

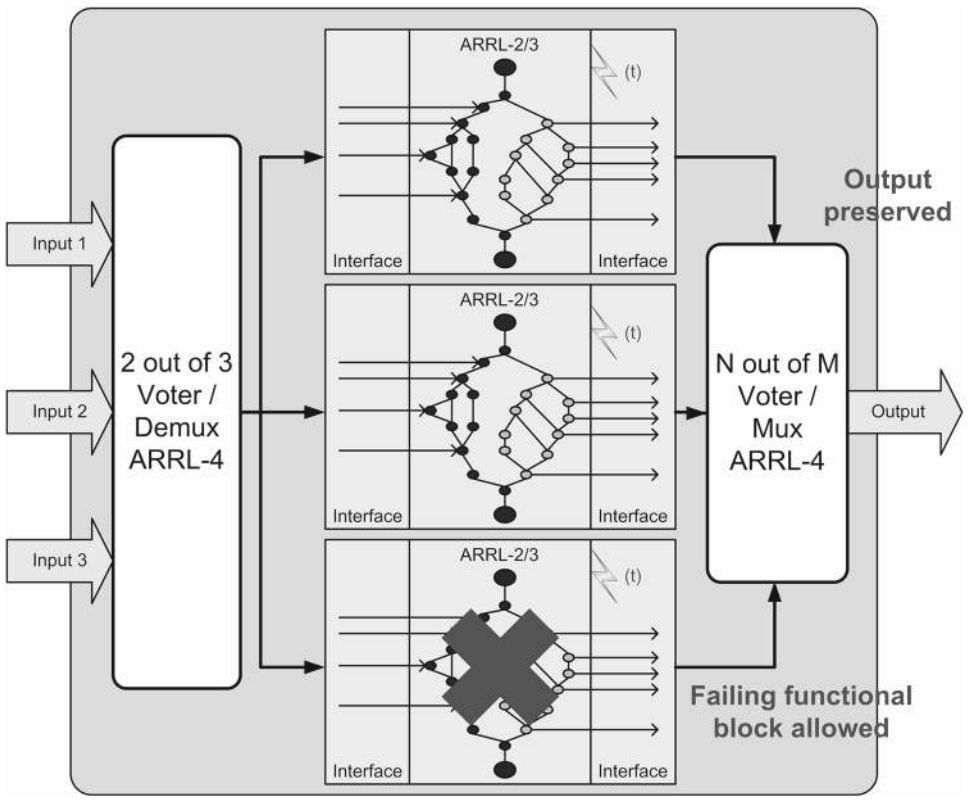

Fig. 6 A generic ARRL-4 component

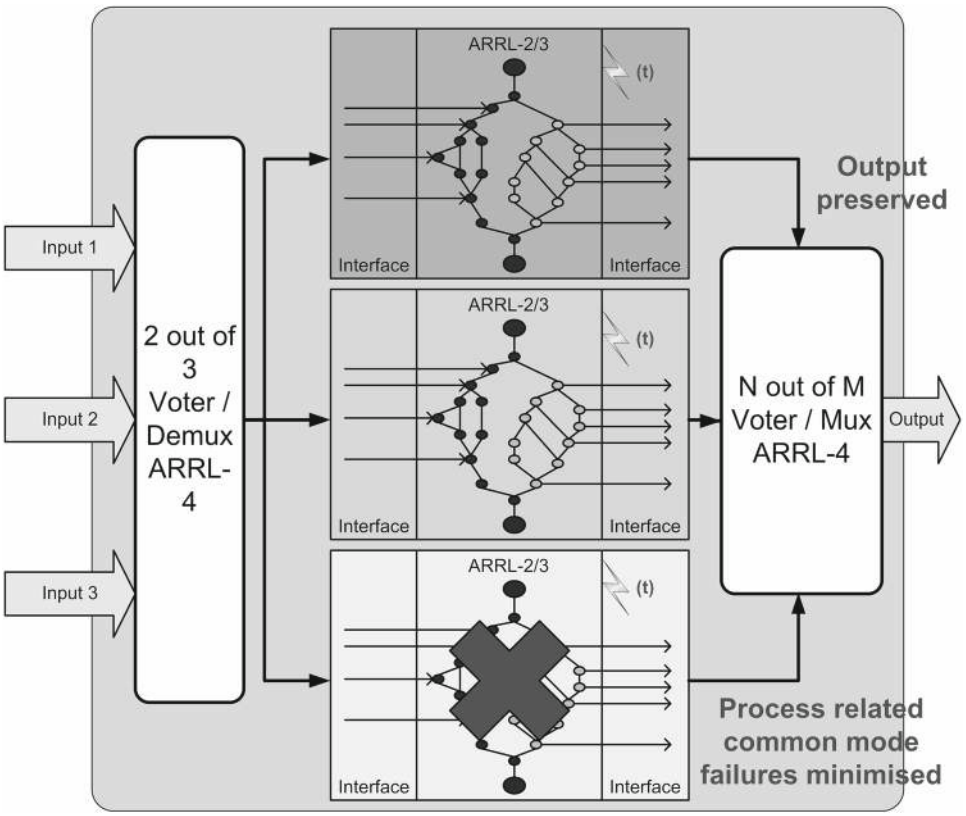

Fig. 7 A generic ARRL-5 component

ARRL-1 component cannot be used as such for safety-critical systems.

5.3 An illustrated ARRL-2 component

An ARRL-2 component (Fig. 4) covers the holes left at the ARRL-1 level. To reach completeness of absence of errors, we first of all assume that the underlying hardware (at the material level) does not introduce any faults from which errors can result. Therefore we speak of "logical correctness" in absence of faults. This level can only be reached if there is formal evidence supporting such a claim. At the hardware level, this means for example extensive design verification, extensive testing and even burn-in of components to find any design or production-related issues. At the software level we could require formal proof that no remaining errors exist. If not practical, formal evidence might also result from "proven in use" arguments whereby stress testing can be mandatory. The latter are weaker arguments than those provided by formal techniques, but even when formal techniques are used, one can never be 100 % sure because even formal models can have errors but they generally increase the confidence significantly. Such errors can further be mitigated by additional process steps (like reviews, continuous integration and validation) but in essence the residual errors should have a probability that is as low as practically feasible so that in practice the component would be considered error-free and hence fully trustworthy, at least if no faults induce errors.

5.4 An illustrated ARRL-3 component

An ARRL-3 component (Fig. 5) inherits first of all the properties of ARRL-2. This means, its behavior is logically correct in absence of faults in relationship to its specifications. ARRL-3 introduces additionally:

- Faults (by default induced by the hardware or by the environment) are detected.
- Faulty input values are remapped to a valid range (e.g. by clamping) whereby a valid range value is one that is part of the logically correct behavior.
- Two processing units are used. These can be identical or dissimilar as long as faults are detected before the component can propagate them as erroneous values to other components.
- Faults induced in the components are detected by comparison at the outputs.
- The output values are kept within a legal range, hence faulty values will not result in an error propagation that can generate errors downstream in the system.

Note that above does not exclude more sophisticated approaches. Certain faults induced in each sub-unit, typi-

cally transient faults, can be locally detected and corrected so that the output remains valid. The second processing unit can also be very different and only act as a monitor (which assumes that faults are independent in time and space). Common mode failures are still a risk.

5.5 An illustrated ARRL-4 component

ARRL-3 components detect failures and prevent error propagation but they result in the system loosing its intended functionality. This is due to the fact that redundancy is too low to reconstruct the correct state of the system. An ARRL-3 component addresses this issue by applying N out of M ($N < M, N >= 2$) voting. This applies as well to the input as to the outputs. This allows to safeguard the functionality at ARRL-3 level and is a crude form of graceful degradation. The solution also assumes independence of faults in the M "channels" and hence most common mode failures are mitigated. This boundary condition implies often that no state information (such as introduced by the power supply) can propagate to another channel.

Note that while the diagram uses a coarse grain representation, some systems apply this principle at the micro-level. For example radiation-hardened processors can be designed to also support Single Event Upsets by applying triplication and voting at the gate level. This does not address all common mode failures (like power supply issues) but often such a component can be classified as an ARRL-4 component (Fig. 6) (implying that in the example the power supply is very trustworthy).

5.6 An illustrated ARRL-5 component

An ARRL-4 component provides continuity in its functionality but can still fail due to residual common mode failures. Most of the residual common mode failures are process related. Typical failures are related to the specifications not being complete or wrong due to misinterpretation. Another class of failures could be time dependent. To mitigate the resulting risks, diversity is used. This can cover using completely different technologies, different teams, applying different algorithms and even using time shifting or using orthogonal placement of the sub-components to reduce the influence of externally induced magnetic fields. In the figure (Fig. 7) this is visualized by using different colors.

This diversity technique is an underlying principle in most safety engineering processes, for example by requiring that tests be done by different people than those who developed the item. It also has as a consequence that such an architecture works with a minimum of asynchronicity, whereby the subcomponents "handshake" (in a time window), which is only possible if the sub-components can be trusted in the sense of ARRL-2 or ARRL-3.

5.7 Rules of composition (non-exhaustive)

A major advantage of the ARRL criterion is that we can now define a simple rule for composing safety-critical systems. We use here an approximate mapping to the different SIL definitions by taking into account the recommended architecture for reaching a certain SIL level.

"A system can only reach a certain SIL level if all its components are at least of the same ARRL level or if they are arranged into a whole that exhibits a higher ARRL level due to the application of a fault tolerant architecture."

The following side-conditions apply:

- The composition rule defines a necessary condition, not a sufficient condition. Application specific layers must also meet the ARRL criterion.
- ARRL-4 components can be composed out of ARRL-3 components using redundancy. This requires an additional ARRL-4 voting component
- ARRL-3 components can be composed using ARRL-2 components (using at least 2 whereby the second instance acts as a monitor).
- All interfaces and interactions need to have the same ARRL level as the components.
- Error propagation is to be prevented. Hence a partitioning architecture (using a distributed hardware and concurrent software architecture) is a must.
- ARRL-5 requires an assessment of the certification of independent development and, when applied to software components, a certified absence of correlated errors.
- A benefit of the approach is that it leaves less room for ad-hoc, often questionable and difficult to verify decompositions of SIL levels. While this might increase the cost, this will likely be cost-efficient over the lifespan of a given technology and reduce the development cost.

Figure 8 illustrates this for a (simplified) 2 out of 3 voter. Note that the crossbar implements also an ARRL-4 architecture.

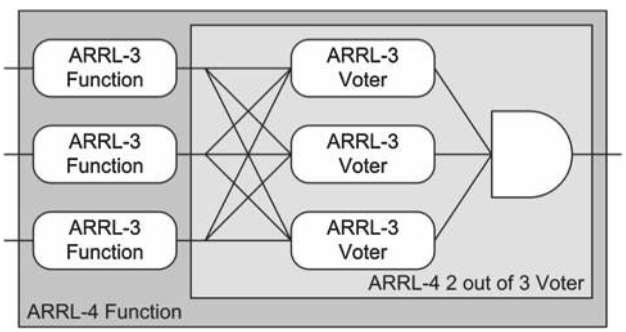

Fig. 8 An AARL_4 2-out-of-3 voter

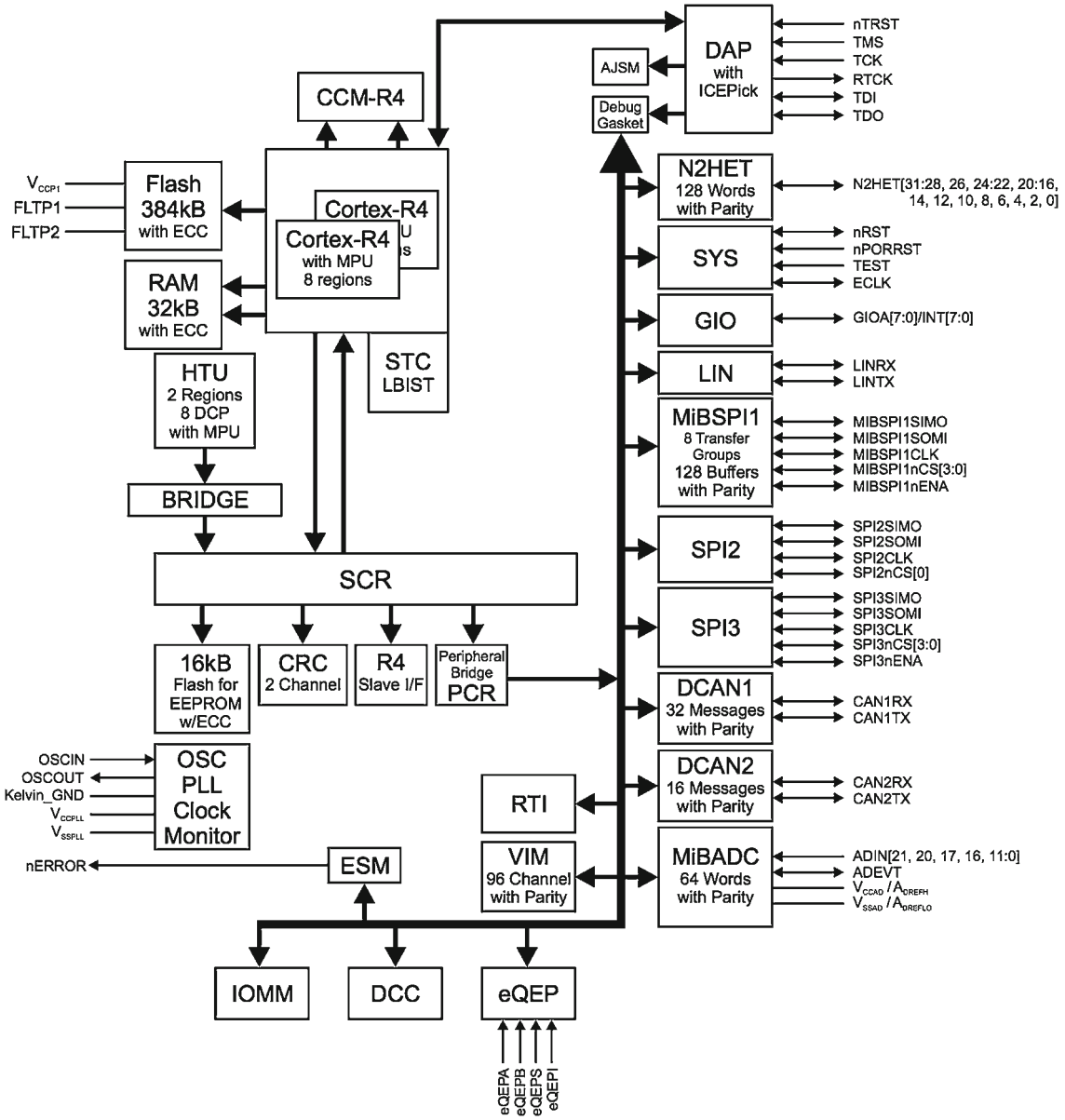

Fig. 9 Texas Instruments' Hercules microcontroller

5.8 The role of formal methods

ARRL-2 introduces the need for formal correctness. This might lead to the conclusions that ARRL-2 makes the use of formal techniques mandatory as well as providing a guarantee of correctness. This view needs further nuance.

In recent years, formal methods have been gaining attention. This is partly driven by the fact (and awareness) that testing and verification can never provide complete coverage of all possible errors, in particular for discrete systems and specifically for software. This is problematic because safety and security issues often concern so-called "corner cases" that do not manifest themselves very often. Formal methods however have the potential to cover all cases either by using formal models checkers (that automatically verify all possible states of the model) or by formal proofs (based on mathematical reasoning). In general we can distinguish a further separation in two domains: the numeral accuracy and stability domain and the event domain whereby the state space itself is verified. Often the same techniques cannot be applied for both.

Practice has shown that using formal methods can greatly increase the trustworthiness of a system or component. Often it will lead to the discovery of logical errors and incomplete assumptions about the system. Another benefit of using formal methods during the design phase is that it helps in finding

cleaner, more orthogonal architectures that have the benefit of less complexity and hence provide a higher level of trustworthiness as well as efficiency [20]. One can therefore be tempted to say that formal methods not only provide correctness (in the sense of the ARRL-2 criterion) but also assist in finding more efficient solutions.

Formal methods are however not sufficient and are certainly not a replacement for testing and verification. Formal methods imply the development of a (more abstract) model and also this model cannot cover all aspects of the system, especially non-functional ones. It might even be incomplete or wrong if based on wrong assumptions (e.g. on how to interpret the system's requirements). Formal methods also suffer from complexity barriers, typically manifested as a state space explosion that makes their use impractical. The latter however is a strong argument for developing a composable architecture that uses small but well proven trustworthy components as advocated by the ARRL criterion. At the same time, the ARRL criterion shows that formal models must also model the additional functionality that each ARRL level requires. This is in line with what John Rushby puts forward in his paper [21] whereby he outlines a formally driven methodology for a safe reuse of components by taking the environment into account.

The other element is that practice has shown that developing a trustworthy system also requires a well-managed engineering process whereby the human factor plays a crucial role [7]. Moreover, processes driven by short iteration cycles whereby each cycle end with a validation or (partial) integration have proven to be more cost-efficient as well as more trustworthy with less residual issues. Formal methods are therefore not something like a miracle cure. Their use is part of a larger process that aims at reaching trustworthiness. The benefit of using formal methods early in the design phase is that it contributes to reducing the state space in an early stage so that the cost and effort of fixing issues that are discovered later in the process is much reduced. In the context of the ARRL criterion they increase the assurance level considerably because of the completeness of the verification, a goal that is only marginally reachable by only testing.

5.9 Applying ARRL on a component

Texas Instruments offers an ARM based microcontroller (MCU) with a specific architecture aimed at supporting embedded safety-critical applications (Fig. 9) [22]. The MCU has many features that support this claim. The most important one is that the ARM CPU adopts an ARRL-3 architecture whereby both CPU cores are lock-stepped. In case of a difference between the two CPUs, the MCU is halted. To mitigate common mode failures a time delay of 2 clock pulses is used and in addition the two cores are rotated with 90° to reduce e.g. electromagnetic disturbances. In addition, Memory Protection Units (MPU) allow the programmer to partition the software in isolated memory blocks. The chip also has quite a number of additional safety (or rather: reliability) features. For example, most memory has error correcting logic to handle bit errors.

At first sight the MCU could be classified as an ARRL-3 component because the processing cores are configured in lockstep mode. However, the chip has also a large number of peripherals in a single instance on the chip. While some have parity bits (but not all), they can most likely be classified as ARRL-2 components on the chip. In addition, the chip has a programmable timer block (that has its own small controller) that is not protected at all from faults. Note that this is deduced from the publicly available documentation. Further information might have an impact on these conclusions.

What can we conclude from this, granted superficial, exercise? First of all, while the MCU core processor can be classified as ARRL-3, most of the peripherals are ARRL-2 or even ARRL-1. Hence, the whole MCU, even better supporting safety-critical applications than most off-the-shelf MCUs, is still an ARRL-2 component, unless one doesn't use some of the peripherals or if the faults are mitigated at the software level. Secondly, ARRL components must carry a contract and the evidence. Even if the documentation supplied by the manufacturer is extensive, it is not in a form that allow a definite conclusion to be drawn. This is in line with the requirements of safety standards, whereby extensive process evidence as well as supporting documentation is required to qualify or certify a system or sub-system.

The example also clearly shows that starting from the ARRL-3 level, it becomes difficult to develop software components in isolation of the hardware it is running on (ARRL-2 level software is assumed to be perfectly error-free in absence of hardware faults). This is due to the fact that additional fault handling at ARRL-3, vs. ARRL-2, is hardware and often application specific. Nevertheless, it is partially possible by strictly specifying the boundary values that are valid for the software component. The errors resulting from hardware faults can then be trapped in the interface layer, that itself can be considered as a software component that often will make use of the underlying hardware support.

5.10 SIL and ARRL are complementary

The ARRL level criterion is not a replacement for the SIL level criterion. It is complementary in the same sense that the HARA and FMEA are complementary (Fig. 10). The HARA is applied top-down whereby the system is considered in its environment including the possible interactions with a user or operator. The goal of the HARA is to find the situations whereby a hazard can result in a safety risk. The outcome is essentially a number of safety measures that must be part of

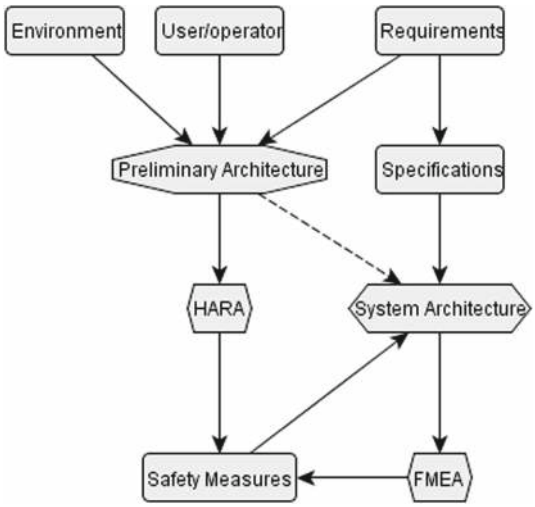

Fig. 10 Correspondence between HARA and FMEA with SIL and ARRL

the system design without necessarily prescribing how these are to be implemented

The FMEA takes a complementary approach after the implementation architecture has been selected. FMEA aims at identifying the faults that are likely to result in errors ultimately resulting in a system failure whereby a safety risk can be encountered. Hence the HARA and FMEA meet in the middle confirming their findings.

By introducing the ARRL criterion we take a first step towards making the process more normative and generic, of course still a tentative step because it will require validation in real test cases. The SIL is a top-level requirement decomposed in normal case requirements (ARRL-1 and -2) and fault case requirements (ARRL-3, -4, -5). From a functional point of view, all ARRL levels provide the same functionality but with different degrees of assurance and hence trustworthiness from the point of view of the user. Concretely, different ARRL levels do not modify the functional requirements and specifications of the components. The normative ARRL requirements result in additional functional specifications and corresponding functional support that assures that faults do not result in the functional specifications to be jeopardized. The non-functional specifications might be impacted as well. For example, the additional functionality will require more resources (e.g. memory, energy and CPU cycles) and is likely to increase the cost price of the system. However, it provides a way to reuse components with lesser efforts from one domain to another in a product family. For example a computer module (specified for compatible environmental conditions) can be reused between different domains. The same applies to software components. However, this requires that the components are more completely specified than it is now often the case. ARRL level components carry a contract and the supporting evidence that they will meet this contract given a specific set of fault conditions. Note that when using formal methods, each of these ARRL levels also requires different formal models. The higher-level ARRL models must model the fault behavior in conjunction with the normal behavior, just like invariants are part of the formal models. By defining a composition rule of ARRL components to achieve a certain level of Safety, we now also define safety in a quasi-domain independent way, simplifying the safety engineering process. Note however that any safety-critical system still has an application specific part that must be developed to meet the same level of ARRL to reach the required SIL.

5.11 An ARRL inspired process flow

We can now also define an ARRL inspired process flow. It is strictly top-down for the requirements engineering part while bottom-up for developing the architecture. The reader should note that such a process is not limited to safety engineering but rather considers this as a special case of systems engineering in general.

It is shown in Table 5, in a simplified way. For simplicity, we merged the ARRL-1 and -2 levels as de facto, ARRL-1 provides very little assurance in terms of safety.

6 Do we need an ARRL-6 and ARRL-7 level?

An ARRL-5 system can be seen as a weak version of a resilient system. While it can survive a major fault, it does so by dropping into an ARRL-4 mode. The next failure is likely catastrophic. However airplanes are also designed as part of a larger system that helps to prevent reaching that state. Continuous built-in-test functions and diagnostics will detect failures before they become a serious issue. Ground crews will be alerted over radio and will be ready to replace the defective part upon arrival at the next airport. We could call this the ARRL-6 level whereby fault escalation is constrained by early diagnostics and monitoring and the presence of a repair process that maintains the operational status at an optimal level. Note that in large systems like server farms and telecommunication networks similar techniques are used. Using monitoring functions and hot-swap capability on each of the 1000's of processing nodes, such a system can reach almost an infinite lifetime (economically speaking). Even the technology can be upgraded without having to shut down the system.

The latter example points us in the direction of what normative ARRL-6 and -7 levels could be. Those are levels whereby the system is seen as a component in a larger system that includes a continuous monitoring and improvement process. The later implies a learning process as well. The aviation industry seems to have reached this maturity level. The

Table 5 An ARRL driven process flow

Phase	ARRL-1, ARRL-2	ARRL-3 additional	ARRL-4 additional	ARRL-5 additional
Requirements capturing	Normal cases test cases	Fault cases (safety and security cases)	Requirements on fault tolerance	Requirements on diversity and independence
Specifications derivation by refinement	Functional and non-functional specifications derived from requirements	Safety and security specifications derived from safety requirements by analysis (HARA). Explicit fail-safe mode	Specifications on selected fault-tolerant architecture	Specifications on selected diversity support
Model building by refinement and specifications mapping	Architectural model. Simulation model. Formal models from ARRL-2 on	Formal models. All models include safety and security support	See ARRL-3. Models include fault-tolerant functionality	See ARRL-4 Heterogeneous models
Model analysis and verification / testing	On normal case architecture and models	Evidence of a fail-safe architecture	Evidence of a fault-tolerant architecture	Evidence of an heterogeneous / design diverse fault-tolerant architecture
Implementation	Manual or code generation	See ARRL-2, code generation recommended	See ARRL-3	See ARRL-4
Integration and validation	Does the ARRL-1, -2 implementation meet SIL-1 or -2 level?	Does the ARRL-3 implementation meet the SIL-3 level?	Does the ARRL-4 implementation meet the SIL-4 level?	Does the ARRL-5 implementation meet the SIL-5 level?

Table 6 ARRL-6 and ARRL-7 definitions

ARRL level	ARRL definition
ARRL-6	The component (or subsystem) is monitored and designed for preventive maintenance whereby a supporting process repairs or replaces defective items while maintaining the functionality and system's services
ARRL-7	The component (or subsystem) is part of a larger "system of systems" that includes a continuous monitoring and improvement process supervised by an independent regulating body
Inheritance rule	The component inherits all properties of any lower level ARRL properties

term maturity is no coincidence, it reminds us of the maturity levels as defined by CMMI levels for an organization. Table 6 summarizes the new ARRL levels whereby we remind the reader that each ARRL level inherits the properties of the lower ARRL levels.

6.1 Beyond ARRL-5: antifragility

Antifragility is a term quote by Taleb [7], mostly in the context of a subjective human social context. He quotes the term to indicate something beyond robustness and resilience that reacts to stressors (and alike) by actually improving its resistance to such stressors. Taking this view in the context of systems engineering we see that such systems already exist. They are distinguished by considering the system as a component in a greater system that includes the operating environment and its continuous processes and all its stakeholders. Further differences are a culture of openness, continuous striving for perfection and the existence of numerous multi-level feedback loops whereby independent authorities guide and steer the system as a whole. The result is a system that evolves towards higher degrees of antifragility. An essential difference with traditional engineering is that the system is continuously being redefined and adapted in an interactive process that aims at increasing the antifragile of the system. As we have seen in the earlier sections, a domain like aviation has a process in place that over the years has resulted in an increasing QoS. Airplanes are above a minimum distance the safest but also the most cost-efficient and energy-efficient way of traveling. As we will see, systems engineering already applies some antifragility principles, but not a strictly normative way as ARRL aims to define.

Applying the concept of anti fragility to the ARRL criterion allows us to define two new levels for the normative ARRL criterion. ARRL-6 indicates a system that preven-

tively seeks to avoid failures by preventive maintenance and repair. ARRL-7 requires a larger process that is capable of not only repairing but also updating the system in a controlled way without disrupting its intended services, unless that is needed. Given the existence of systems with such (partial) properties, it is not clear whether the use of the neologism "antifragile" is justified to replace reliability and resilience, even if it indicates a clear qualitative and distinctive level. This will need further study.

The normative ARRL levels describe as the name says, levels of reliability and resilience. They approach the notion of graceful degradation by redundancy but assuming that in absence of faults the system components can be considered as error-free. The additional functionality and redundancy (that is also error-free) is to be seen as an architectural or process level improvement. But in all cases, contrary to the antifragility notion, the system will not gain in resilience or reliability. It can merely postpone catastrophic failures while maintaining temporarily the intended services. It does this by assuming that all types of faults can be anticipated, which would be the state of the art in engineering. Of course, in practice all faults can't be anticipated and therefore an additional layer is needed to deal with them. The proposed scheme introduces already two concepts that are essential to take it a step further. Firstly, there is redundancy in architecture and process and secondly, there is a monitoring function that acts by reconfiguring the system upon detecting a fault.

6.2 Antifragility assumptions

So, how can a system become "better" when subjected to faults? As we introduce a metric as a goal, we must somehow measure and introduce feedback loops. If we extrapolate and scale up, this assumes that the system has a type of self-model of itself that it can use to compare its current status with a reference goal. Hence, either the designer must encapsulate this model within the system or the model is external and becomes part of the system. If we consider systems that include their self-model from the start, then clearly becoming a "better" system has its limits, the limit being the designers's idea at the moment of conception. While there are systems that evolve to reach a better optimum (think about neural networks or genetic algorithms), these systems evolve towards a limit value. In other words they do not evolve, they converge.

If on the other hand we expand the system as in Fig. 1, then the system can evolve. It can evolve and improve because we consider its environment and all its stakeholders of which the users as part of the system. They continuously provide information on the system's performance and take measures to improve upon it. It also means that the engineering process doesn't stop when the system has been put to use for the first time. It actually never ends because the experience is transferred to newer designs.

There are numerous examples of antifragile systems already at work, perhaps not perfect all the time though most of the time. A prime example is the aviation industry that demonstrates by its yearly decreasing number of fatalities and increasing quality of service that it meets the criterion of antifragility. Moreover, it is a commercial success. So let's examine some of its properties and extract the general principles, as reflected in the aviation standards and practice [8].

6.3 Some industries are antifragile by design

To remain synoptic, we will list a few key principles of the aviation industry and derive from them key generic principles which apply to other systems and provide them with antifragile properties.

Table 7 can also be related to many other domains that have a significant societal importance. Think about sectors like medical devices, railway, automotive, telecommunications, internet, nuclear, etc. They all have formalized safety standards which must be adhered to because when failing they have a high impact at socio-economic level.

At the same time, systems like railway that are confined by national regulations clearly have a higher challenge to continue delivering their services at a high level. As a counter example we can take a look at the automotive sector. Many more people are killed yearly in traffic than in airplanes, even if cars today are stuffed with safety functions. In the next section we will explore this more in detail.

Deducting some general properties out of the table Table 7, we can see that systems that could be termed antifragile are first of all not new. Many systems have antifragile properties. Often they can be considered as complex (as there are many components in the system) but they remain resilient and antifragile by adopting a few fundamental rules:

1. Openness: all service critical information is shared and public.
2. Constant feedback loops between all stakeholders at several different levels.
3. Independent supervising authorities.
4. The core components are designed at ARRL-4 and ARRL-5 levels, i.e. fault tolerant.

6.4 Automated traffic as an antifragile ARRL-7 system

As we discussed earlier [23–26], the automotive sector does not yet meet the highest ARRL levels neither in the safety standards (like IEC-26262) [2] and nor in reality. 1000 more people are killed in cars than in airplanes worldwide and even a larger number survive with disabilities.[11,12,27]. The main reason is not that cars are unsafe by design (although fault tolerance is not supported) but because the vehicles are part of a much larger traffic system that is largely an ad-hoc

Table 7 Generic properties derived from observing the avionic sector

Aviation specific	Generic property
The industry has a long track record	The domain has undergone many technological changes whereby an extensive knowledge was built up
Development of systems follows a rigorous, quantifiable, certifiable process, that is widely published and adopted	The process is open and reflects the past experience and is certified by an independent external authority
Certification requirements foster developing "minimal" implementations that still meet the operational requirements	Systems are designed to be transparent and simple, focusing on the must-haves and not on the nice to haves
Airplanes are designed to be 100 % safe and to be operated in 100 % safe conditions	The domain has a goal of perfection. Any deviation is considered a failure that must be corrected. By design the system, its components and operating procedures aim at absence of service and safety degradation
Any failure is reported in a public database and thoroughly analyzed. After analysis, immediate action can be mandated to rectify the issues to prevent future mishaps	Any issue is seen as a valuable source of information to improve processes and systems
Airplanes are operated as part of a larger worldwide system that involves legal authorities, the operators, the manufactures, the public and supervising independent authorities.	A (sub)system is not seen in isolation but in its complete socio-economic context. This larger system is self-regulating but supervised and controlled by an independent authority
Airplanes have a long life time and undergo mid-life updates to maintain their serviceability	The focus is on the service delivered and not on the system as a final product
Fault conditions are preventively monitored. The system is fault tolerant through redundancy, immediate repair and preventive maintenance	A process is in place that maintains the state of the system at a high service level without disrupting the services provided

system. Would it be feasible to reach a similar ARRL level as in the aviation industry? What needs to change? Can this be done by allowing autonomous driving?

A first observation is that the vehicle as a component now needs to reach ARRL-4, even ARRL-5 and ARRL-6 levels. If we automate traffic, following design parameters become crucial:

- The margin for driving errors will greatly decrease. Vehicles already operate in very dynamic conditions whereby seconds and centimeters make the difference between an accident and not an accident. With automated driving, bumper to bumper driving at high speed will likely be the norm.
- The driver might be a back-up solution to take over when systems fail, but he is unlikely to be trained well enough and therefore to react in time (seconds).
- A failing vehicle can generate a serious avalanche effect whereby many vehicles become involved and the traffic system can be seriously disrupted.

Hence, vehicles need to be fault tolerant. First of all they have to constantly monitor and diagnose the vehicle components to pro-actively prevent the failing of subsystems and secondly when a failure occurs the function must be maintained allowing to apply repair in a short interval.

A second observation is that the automated vehicle will likely constantly communicate with other vehicles and with the traffic infrastructure. New vehicles start to have this capability today as well, but with automated vehicles this functionality must be guaranteed at all times as disruption of the service can be catastrophic.

A third observation is that the current road infrastructure is likely too complex to allow automated driving in an economical way. While research vehicles have been demonstrated the capability to drive on unplanned complex roads, the question is whether this is the most economical and trustworthy solution.

Automated traffic can be analyzed in a deeper way. Most likely, worldwide standardization will be needed and more openness on when things fail. Most likely, fully automated driving providing very dense traffic at high speed will require dedicated highways, whereas on secondary roads the system will be more a planning and obstacle avoidance assistance system to the driver. One can even ask if we should still speak of vehicles. The final functionality is mobility and transport. For the next generation, cars and trucks as we know them today might not be the solution. A much more modular and scalable, yet automated, transport module that can operate off-road and on standardized auto-highways is more likely the outcome. Users will likely not own such a module but

rent it when needed whereby operators will be responsible for keeping it functioning and improving it without disrupting the service. Independent authorities will supervise and provide an even playing field. Openness, communication and feedback loops at all levels will give it the antifragility property that we already enjoy in aviation.

6.5 Is there an ARRL-8 level and higher?

One can ask the question whether we can define additional ARRL levels. ARRL levels 0 to 7 are clearly defined in the context of (traditional) systems engineering whereby humans are important agents in the required processes to reach these levels. One could say that such a system as shown in Fig. 1 is self-adaptive. However the antifragile properties (even when only partially fulfilled) are designed in and require conscious and deliberate actions to maintain the ARRL level. If we look at biological systems we can see that such systems evolve without the intervention of external agents (except when they stress the biological system). Evolution as such has reached a level whereby the "architecture" is self-adaptive and redundant without the need for conscious and deliberate actions. We could call this the ARRL-8 level.

When considering bio- and genetic engineering, we can see that we could take it a step further. Genetic engineering (and that includes early breeding techniques) involves human intervention in ARRL-8 level systems. The boundaries however become fuzzy. One could consider this as an ARRL-9 level but also as an ARRL-7 level using biological components. This raises interesting philosophical and ethical questions that requires a deeper understanding on how genetic building blocks really work. This topic requires further study and is not within the scope of this paper.

7 Conclusion and future work

This paper analyzed the concept of safety integrity level (SIL) and put it in a wider perspective of quality of service and trustworthiness. These concepts are more generic and express the top-level requirements of a system in the perspective of a prospective user. We have discussed some weaknesses in the SIL concept, mainly its probabilistic system view whereas engineering is often based on composition using components or sub-systems. A new concept called ARRL was introduced defining a normative criterion for components and their interactions. However, it was shown that SIL and ARRL are complementary. An ARRL enabled process flow was defined. It has the advantage that it better separates the additional safety functions from the normal use case support than the traditional more monolithic approach.

As to future work, the concept will further be validated and applied in the context of safety-critical applications. This will help in deepening the criterion and allowing it to be used for defining contract carrying components. Issues that need further elaboration are for example:

- How can an ARRL level as a design goal be refined into sub goals?
- When is a contract complete and sufficient to certify that a given ARRL level has been reached?
- How can the component's contract and evidence be provided in an application domain independent way?
- What is the impact on the safety/systems engineering process?
- What is the impact on the system architecture?

Another important issue is analyzing how the composition of a system by using ARRL qualified components results in emerging properties that can result in a safety-critical state. The underlying assumption here is that a system can already be in a critical state, to be seen as a collection of erroneous states present in its components, before an event can trigger a catastrophic state for the whole system. While this aspect was briefly touched by requiring partitioning support and corresponding ARRL levels for the interaction components, this requires further attention. An interesting question is for example if such a critical state can be detected before it results in a catastrophic event.

At Altreonic, work is currently in progress to apply the ARRL criterion on the internally developed real-time operating system OpenComRTOS [20]. While formally developed and a lot of supporting evidence is available, still missing supporting evidence has been identified. This was greatly facilitated by the use of the GoedelWorks portal that allows importing a software repository and its supporting documents. Most of the issues identified are related to the process followed. This indicates, as it is the case in most safety engineering projects, that an ARRL driven development must take the normative criteria into account from the very beginning. If so, the supporting evidence, generated and stored in the GoedelWorks repository, will provide a "qualification" package for the product developed. This is similar to the qualification requirements for externally procured components and subsystems as found in most safety standards. The difference is that an ARRL qualified component will be much more domain independent, a design goal that is also fulfilled by the GoedelWorks generic metamodel.

Nevertheless, we believe that the ARRL criterion, being normative, is a promising approach for achieving safety across different domains and systems in a product family by composing qualified trustworthy components. At the same time it puts forward that the specification of a component with its contract and supporting evidence is a complex under-

taking but in line with the sometimes unspoken assumptions one finds back in safety and systems engineering texts.

This is work in progress. Feedback and contributions are welcome.

References

1. Ledinot E, Astruc JM, Blanquart JP, Baufreton P, Boulanger JL, Delseny H, Gassino J, Ladier G, Leeman M, Machrouh J, Qur P, Ricque B (2012) A cross-domain comparison of software development assurance standards. In: ERTS^{2}2012. http://web1.see.asso.fr/erts2012/Site/0P2RUC89/1A-3.pdf
2. Functional Safety and IEC 61508. http://www.iec.ch/functionalsafety/
3. Automotive safety integrity level. http://www.flandersdrive.be/sites/default/files/publicaties/ASIL%20-%20Public%20Results.pdf
4. Trustworthy systems engineering with goedelworks. http://www.altreonic.com/sites/default/files/SE%20with%20GoedelWorks%203_0.pdf
5. Open platform for evolutionary certification of safety-critical systems. http://www.opencoss-project.eu/
6. Gsn (goal structuring notation). http://www.goalstructuringnotation.info/
7. Taleb NN (2012) Antifragile. Random house, things that gain from disorder
8. ISO, Road vehicles—Functional safety, ISO 26262, part 1–10, International standard under publication, Geneva, 2011
9. CENELEC, EN50128—Railway applications-Communication Signaling and Processing Systems-Software for Railway Control and Protection Systems, European standard under publication, Brussels, 2011
10. Special C. of RTCA, DO-178C Software Considerations in Airborne Systems and Equipment Certification (2011)
11. Aircraft crashes record office (acro). http://www.baaa-acro.com/
12. Global status report onroad safety (2013) Supporting a decade of action. Tech rep, World Health Organisation. http://www.who.int/iris/bitstream/10665/78256/1/9789241564564_eng.pdf
13. Goel L, Tyagi VK (1993) A two unit series system with correlated failures and repairs. Microelectron Reliab 33(14):2165–2169. doi:10.1016/0026-2714(93)90010-V. http://www.sciencedirect.com/science/article/pii/002627149390010V
14. Wikipedia: Boeing 787 dreamliner battery problems—wikipedia, the free encyclopedia (2015). https://en.wikipedia.org/w/index.php?title=Boeing_787_Dreamliner_battery_problems&oldid=672415823. Accessed 28 July 2015
15. RTCA (2005) DO-297 integrated modular avionics (IMA) development guidance and certification considerations
16. Florio VD, Blondia C (2008) On the requirements of new software development. Int J Bus Intell Data Min 3(3):330–349. doi:10.1504/IJBIDM.2008.022138
17. Leveson NG (2012) Engineering a safer world. MIT Press
18. De Florio V (2015) On resilient behaviors in computational systems and environments. J Reliab Intell Environ 1(1):33–46. doi:10.1007/s40860-015-0002-6
19. De Florio V (2010) Software assumptions failure tolerance: role, strategies, and visions. In: Casimiro A, de Lemos R, Gacek C (eds) Architecting dependable systems VII. Lecture notes in computer science, vol. 6420. Springer, Berlin Heidelberg, pp 249–272. doi:10.1007/978-3-642-17245-8_11
20. Verhulst E, Boute R, Faria J, Sputh B, Mezhuyev V (2011) Formal development of a network-centric RTOS. Springer, New York
21. Rushby J (2012) Composing safe systems. In: Arbab F, lveczky P (eds) Formal aspects of component software. Lecture notes in computer science, vol 7253. Springer, Berlin Heidelberg, pp 3–11. doi:10.1007/978-3-642-35743-5_2
22. Texas instruments (2013) RM42x 16/32-Bit RISC flash microcontroller technical reference manual. http://www.ti.com/lit/ug/spnu516a/spnu516a.pdf
23. From safety integrity level to assured reliability and resilience level for compositional safety critical systems. http://www.altreonic.com/sites/default/files/Altreonic_ARRL_DRAFT_WIP011113.pdf
24. Verhulst E, Sputh B (2013) ARRL: a criterion for compositional safety and systems engineering. A normative approach to specifying components. In: 2013 IEEE international symposium on software reliability engineering workshops (ISSREW), pp 37–44. doi:10.1109/ISSREW.2013.6688861
25. Verhulst E, de la Vara JL, Sputh B, de Florio V (2013) ARRL: a novel criterion for composable safety and systems engineering. In: SafeComp/SASSUR workshop. Toulouse
26. Verhulst E, de la Vara JL, Sputh B, de Florio V (2013) From safety integrity level to assured reliability and resilience level for composable safety critical systems. In: ICSSEA'13 (2013)
27. European Commission, Statistics—accidents data. http://ec.europa.eu/transport/road_safety/specialist/statistics/index_en.htm