ISO26262 SEooC Compliance of a ROS Based Architecture

ISMAEL ETXEBERRIA-AGIRIANO[1], XABIER LARRUCEA[1,2], PABLO GONZALEZ-NALDA[1],
MARI CARMEN OTERO[1], ISIDRO CALVO[3]
[1] Department of Computer Languages and Systems
[2] ICT – European Software Institute Division - Tecnalia
[3] Department of Automatic Control and Systems
[1,3] University College of Engineering - University of the Basque Country (UPV/EHU)
[1,3] Nieves Cano, 12. 01006 Vitoria-Gasteiz
[2] Parque tecnológico de Bizkaia. Calle Geldo Edificio 700. Derio (Bizkaia)
[1,2,3] SPAIN
{ismael.etxeberria,pablo.gonzalez,mariacarmen.otero,isidro.calvo}@ehu.eus
xabier.larrucea@tecnalia.com

Abstract: - Robot Operating System (ROS) begins to be used in automotive industry as a component to be adapted and deployed in cars. However, its use varies according to a set of parameters, and its reliability depends on these values, and usage models. This paper proposes a certification approach based on evidences for a ROS based architecture aligned with the ISO26262 and its Safety Element out of Context (SEooC) component definition. This ROS based architecture is being tested in order to identify characteristics and thresholds to be used during the whole development life cycle, safety case definition and especially during the certification phase. Finally we have outlined an ISO26262 based certification process for this kind of component.

Key-Words: - ROS, reliability, ISO26262, SEooC, Certification, Safety Case.

1 Introduction

Nowadays the so called autonomous vehicles are becoming popular. Their core functionalities are relying on a dvanced software and hardware capabilities. Press and scientific literature are reporting experiences on autonomous cars [1]. Some car manufacturers are using the Robot Operating System (ROS) such as BMW[1]. ROS is a set of open source software libraries for building robot applications. Literature reports experiences applying robot operating systems in several domains [2] [3] [4], and even in the automotive industry [5] [6].
Safety certification is currently a hot topic for industry [7] especially in safety critical software systems [8]. Certification of electronic components in cars, and in particular their embedded software, is a major goal that the automotive industry tries to reach [9]. In this sense, ISO26262 [10] is becoming the reference model for the automotive industry [11] covering the whole development life cycle. Some tools have been developed for supporting this life cycle such as OASIS [12] or OpenCert [13]. Safety certification relies on the demonstration that a software system is acceptably safe by appealing to the satisfaction of a set of objectives that the safety standards require for compliance [14]. In order to successfully pass a ce rtification process an organization has to demonstrate that its development processes and their resulting products fulfil a set of relevant standards or their reference models [15]. A safety case composed by arguments and evidences [16] provides enough confidence that a given system complies with these requirements [17]. There are experiences relating safety cases in compliance with the ISO26262 [18]. Benefits and weaknesses of using safety cases ar e discussed in [19]. CertWare [20] is also aligned with the certification of safety critical domains, and it proposes the use of safety cases [7].
A ROS based architecture can be defined for the automotive domain. Robotic systems can handle several functionalities and ROS can be an open source specific component. In fact ROS can be used as a Safety Element out of Context (SEooC) as defined by the ISO26262 [10] part 10, and reused in several systems. However in order to utilise this component we need to identify requirements and design assumptions as it is highlighted in Fig 1 and described by ISO26262 [10]. These assumptions impact on t he SEooC requirements and design.

[1] http://www.ros.org

Therefore we need to characterize this ROS based component, such as timing constraints [21], in order to use it in the automotive industry and to fulfil ISO26262 requirements. As the ROS based architecture is flexible there are some parameters which can be modified producing different operational profiles [22].

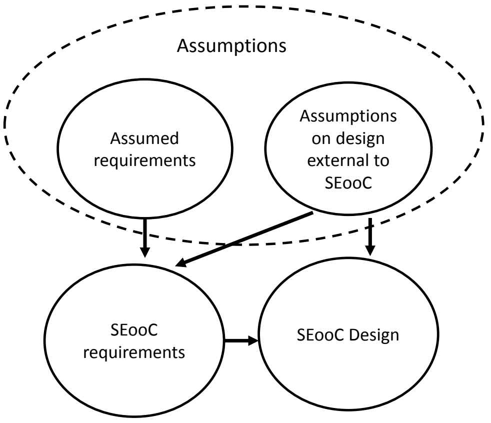


Fig 1: Relationships between assumptions and SEooC [10]

This paper proposes a certification approach based on evidences for a ROS based architecture aligned with the ISO26262 and its SEooC component definition. This ROS based architecture is tested in order to identify characteristics and thresholds to be used during the whole development life cycle, safety case definition and especially during the certification phase. Finally we have outlined an ISO26262 based certification process for this kind of component. In fact this component can be parameterized depending on a set of ROS based architecture aspects which can impact on their Automotive Safety Integrity Level (ASIL).

This paper deals with the following research questions (RQ):

- RQ1: What is the timing behaviour of a ROS based architecture? Basically this component includes sensor, controller and actuator. We need to understand how these elements interact among them, identifying what thresholds are acceptable. In this sense we need to define an operational profile for our ROS based architecture.
- RQ2: What are the safety aspects that should be taken into account during the safety case definition? During a safety analysis of a ROS component treated as a SEooC we need to identify assumptions and arguments to support and reason that a specific instantiation of a ROS based architecture is acceptably safe.
- RQ3: What aspects should be highlighted during a certification process for a ROS based architecture? We need to understand and clarify what evidences are the most relevant for an ISO26262 based certification process.

The remainder of this paper is structured as follows. First a background analysis is provided. Then we define our ROS based architecture to be used in automotive industry. Next we describe the certification process for our ROS component, and the main results stemming from these experiments. After discussing the major results and research questions we summarize our paper.

2 Background

This section analyses the background in various significant aspects in this context: SEooC within an ISO26262 certification, Software Reliability Engineering and Certification, ROS and Safety Cases.

2.1 ISO26262 certification and SEooC

The latest version of ISO26262 [10] was released in 2011 as a standard covering all activities related to functional safety in the context of road vehicles. The SEooC concept (Safety Element out of Context) is defined in part “10 Guideline on ISO26262”.

Basically this standard supports the whole development lifecycle phases. All these phases define product development best practices for system, hardware and software levels. It is widely known that automotive industry does not require a certification based on the ISO26262, but there are some literature referring to a certification process [23] and it is becoming a reference model for the automotive industry. In addition the emergence of autonomous vehicles will increase the relevance of this certification. In our context we focus on software component certification [24], and specifically the so-called SEooC components including hardware and software aspects. Concerning the software aspect, ISO26262 states the specification of the software safety requirements considering the timing constraints among others. All these requirements impact on a wide set of ISO26262 clauses such as specification of software safety requirements, software unit design and implementation, and software unit testing.

Jeff Voas defined a certification process for off-the-shelf components [25] which is based on the analysis of quality characteristics, considering components as black boxes. A component is considered to be certifiable if its quality characteristics are met. Voas' approach does not define what high quality [25] means, and our approach extends his approach with our ROS based architecture. He also considers wrappers to limit its component's behaviour, and the analysis of its operational system which can also affect the component behaviour.

2.2 Software reliability engineering

Musa and Everett defined Software-reliability engineering as the applied science of predicting, measuring, and managing the reliability of software based systems to maximize customer satisfaction [26]. Nowadays there are software based systems such as R OS providing functionalities to critical systems for its appropriate functioning. Software has a r elevant role on these systems. In fact, Software Engineering as a d iscipline is a cornerstone in the development of software based systems. A myriad of aspects are taking relevance in these scenarios such as debugging, early error detection, fast recovery, long term support, dynamic and static analyses, and evolution. One key characteristic is that they must be reliable to some extent. Otherwise the resulting products will not satisfy customer needs, and will have an impact into customer satisfaction.

The theory of software reliability was defined by Musa [27], and several approaches are grounded on this theory such as Software Reliability Growth Models (SRGM). In this sense, according to [28], there are two types of models: black box and white box models.

(1) Black-box Software Reliability Models assume systems as monolithic systems.

(2) White-box Software Reliability Models provide a detailed view of the systems.

Basically Software reliability growth models are used for fault prevention, fault removal, fault tolerance and fault/failure forecasting [28]. Black box models are usually applied during these testing phases which can be considered too late in the development process [29].

2.3 Software reliability certification

Basic steps are defined for implementing software reliability such as [28] where the main steps are: list related systems, implement operational profiles, define necessary reliability, prepare for test, execute test and guide test. A similar approach is the Software Reliability Engineering Process [22] where once the reliability objective is determined and the operational profile is developed, we proceed with the software testing, collection of failure data, and so on. C oncerning certification, certifying the reliability of software is not easy task, and all these approaches agree on the fact that we need to set up the operational profiles. In fact, a certification process will take into account these operational profiles as a basis. In this context safety evidences [30] are used to support certification processes, and they are a co rnerstone in software component certification [24] as well. Wholin and Regnell defined a reliability certification of software components for different usage profiles [31] (see Fig 2). In our context we identify and characterize a usage profile for a SEooC component as well as its reliability. This is needed because during the certification process the auditor/evaluator checks these characteristics in the resulting product by means of measurable attributes.

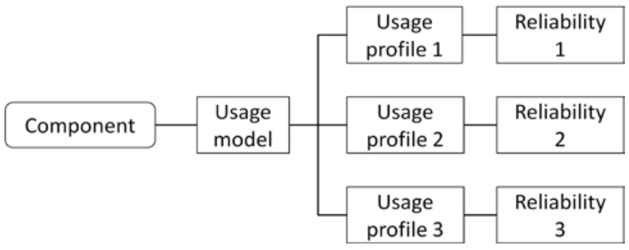


Fig 2: Component, usage model, usage profile and its related reliability [31]

2.4 Robotic Operating System (ROS)

Robotics software has been chronically facing problems in industry and academy due to the lack of standardization, interoperability and reuse of software libraries [32]. The most relevant problems that prevented the robotics community from producing a h ealthy software ecosystem were: (1) lack of code reuse; (2) higher needs of integration of components and; (3) finding the appropriate trade-off between efficiency and robustness. As a solution, free software and open source software (FOS) initiatives such as the Robot Operating System (ROS) initiative were promoted [32].

In particular, ROS [33] provides operating system-like tools and package tools. ROS defines different entities including nodes, message topics and services. Nodes are processes or software modules that can communicate with other nodes by passing simple messages (or data structures) by means of publisher/subscriber mechanisms on t op of TCP or UDP. In ROS a service is modelled as a p air of messages, one for request and another for reply. ROS has several client libraries implemented in different languages such as C++, Python, Octave or Java in order to create ROS applications. Its major advantage is code reuse and sharing [34].

ROS has been successfully used in different kinds of robots such as autonomous guided vehicles [35] or even in the automotive industry. For example, ROS

[5] is proposed to support the Co-Pilot system at highly automated vehicles; the driver should take over the control within a certain time constraint when the system requests it, otherwise the system pulls over the car safely. ROS is also used [35] for establishing a Collision Avoidance system for Autonomous Driving tasks.

2.5 Safety Cases

There is an existing debate whether safety cases are enough to provide confidence in order to consider a system safe [19]. Together with other authors [6] we consider safety cases as a useful way to provide enough confidence using structured arguments and evidences [36] in safety critical applications [37]. In our approach Safety Cases are used for gathering primary assets during certification process.

Safety cases have been applied to several domains such as avionics [38] [39]. The OMG is devoting efforts with a special task force[2] and a dependability assurance framework[3]. They have released a metamodel for representing assurance cases, the so-called Structured Assurance Case Metamodel (SACM).

3 The Architecture of a SeooC for certification testing

This section briefly describes the ROS architecture used and the experimentation carried out.

3.1 ROS Description: through an operational profile

Software architectures are needed in order to improve the software engineering process by connecting components which allow code reuse while keeping an appropriate trade-off between efficiency and robustness. These architectures should be properly certified according to the ISO 26262 standard.

Our component wrapper implemented with ROS technology is aimed at hiding both sensors and actuators by means of a two level architecture:

1. Advanced Level Processing (ALP), implemented on a higher processing device with higher computation and connectivity capabilities
2. Low Level Processing (LLP), implemented on devices with limited capabilities

So, while ALP nodes are typically communicate to similar nodes by means of standard interfaces, such as AUTOSAR interfaces, communication between APL and LLP nodes can use proprietary mechanisms. ROS is adopted in order to avoid low level programming details. For the sake of testing this architecture is implemented using two different and easily available devices: A Raspberry Pi board as ALP node, which processes the data acquired from an LLP node, implemented with an Arduino Board, both linked via an USB serial connection. Fig 3 provides a UML diagram of this architecture. Fig 4 provides a detailed representation of the implementation of this architecture, which is deployed in a vehicle as shown in Fig 5.

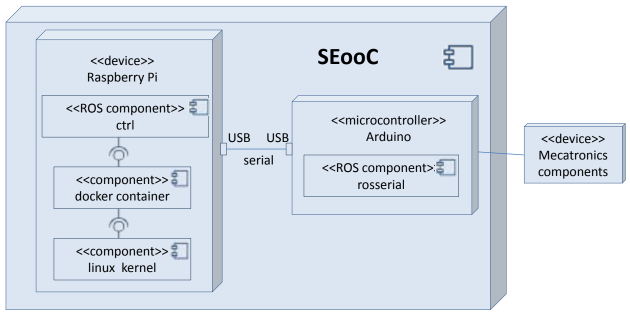


Fig 3: UML Deployment Diagram showing the main elements in our SEooC

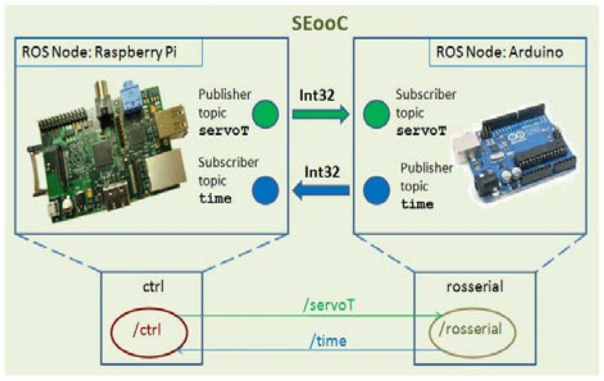


Fig 4: Implementation of the architecture with Raspberry Pi & Arduino

3.2 Architecture used in the experiments

Since this architecture is aimed at building components that could be connected into a broader system, typically for measuring or actuation purposes, they must be represented externally by a limited set of chosen parameters. In order to ease the integration of the component it is of key importance that these parameters represent only the most relevant aspects of the component behaviour, while wrapping to a maximum the characteristics of the internal off-the-shelf components involved, i.e. ROS framework, devices at ALP and LLP layers. The selected parameters are:

1. Period: It describes the time interval, in milliseconds, of communication between the ALP (Raspberry Pi) and LLP (Arduino) nodes.

[2] http://sysa.omg.org/

[3] http://www.omg.org/hot-topics/cdss.htm

2. Granularity: It represents a m essage delivery update frequency parameter related to the ROS message-passing mechanism. It is expressed in kHz.
3. Queue: ROS follows the Publisher/Subscriber paradigm and stores messages in fixed length queues. This parameter is the internal size of these queues.

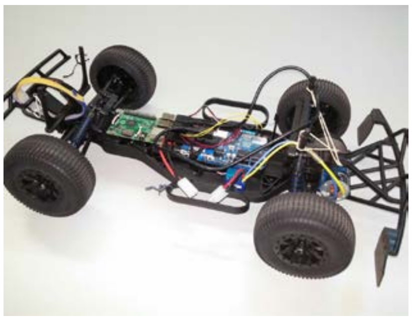


Fig 5: Our Raspberry Pi & Arduino in an autonomous car prototype

4 Certifying a SEooC component

The certification process for our ROS based component is based on [25] [31] [40]. The main activities of the certification process are described as follows (Fig 6) [40]:

1. Scope [40]: this activity is mainly related to the scope definition of our certification process. In fact, our scope is focused on ROS, and on t he Safety Element out of Context based on ISO26262.

2. Primary documentation [40]: this activity is focused on providing the primary certification documentation such as Safety Requirements Specifications, Safety Architecture and Safety Cases [41]. Our focus is to highlight the main argumentations used for building a safety case for a ROS based architecture and ISO26262 SEooC definition used for certification purposes [42].

3. Assessment [40]: This activity is structured in three sub-activities:

a. Component usage model definition: the use of a component can vary its reliability based on a set of properties that are identified for our ROS architecture [31].
b. Measurable attributes: The aforementioned properties are the attributes which are going to be measured.
c. Results: analysis of the results in order to determine whether our ROS component behaviour is relevant.

4. Report & Certificate [40]: basically this activity is a su mmary of the main results of our ROS component with a set of usage models proposed for its integration with other systems.

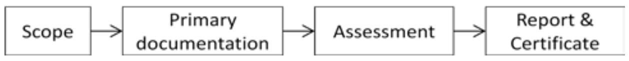


Fig 6: Certification process [40]

Each activity of this process is described in the following sub-sections.

4.1 Scope

This first activity is focused on defining the main scope of the certification. In this sense, it is required to analyze the steps for developing a SEooC based on ISO26262:

1. Assumptions on the scope of SEooC: purpose, boundaries and functionalities for our component are identified.
2. Assumption on Functional Safety Requirement of the SEooC: functional safety requirements for our ROS component are taken into account.
3. Execution of SEooC development: this step is devoted to the whole development of the component.
4. Identification of work products: these work products validate the assumed functional safety requirements, and assumptions are met.
5. SEooC integration: SEooC assumptions are verified, including ASIL capability, and the assumed safety requirements are correctly integrated with the rest of the system.

These assumptions and boundaries characterize our ROS architecture, and measurable attributes are identified. We are therefore going to focus just on identifying them. According to [31], a component has different usage profiles for different reliabilities. In fact we do not know the most appropriate usage profile for our ROS architecture. Consequently, we need to identify these boundaries which help us during the assumptions definition. We have defined a set of experiments (tests) based on by combining three parameters of our ROS based architecture: (1) Time between messages (period), (2) Granularity and (3) Queue size. These elements modify the behaviour of our architecture, and accordingly they modify its reliability. Each experiment or test modifies one parameter of the architecture (Table 1). These parameters characterize the behaviour of our SEooC. It is under designer's responsibility to accept or to decline a specific configuration according to the requirements of a sp ecific application.

4.2 Primary documentation

One relevant aspect during a certification process is the gathering of the primary assets during certification. Clear arguments are the basis for safety certification [41] [42], and they help us to structure primary documentation. All this information is represented by a safety case which is used to demonstrate by argumentations and evidences that our ROS based component is safe and it conforms to the ISO26262. Then this goal is split into several sub-goals which fulfil some of the ISO26262 clauses which are traditionally used in these scenarios. Apart from the hazard analysis, a set of goals related to SEooC are identified such as the design which takes into account all safety requirements. Measures are identified to show the correct implementation of safety requirements. The details of the documentation containing safety cases and our representation using GSN notation is considered out of the scope of this paper.

4.3 Assessment

Our assessment process (Fig 7) is used to determine what ROS aspects are going to be represented and tested for certifications purposes. The following sections describe in detail the activities of this process.

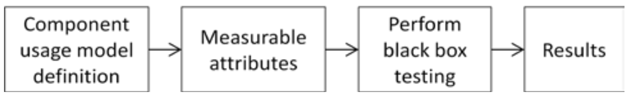


Fig 7: Assessment process

4.3.1 Component usage model definition

Our certification approach is aligned with ISO26262 and the SEooC definition (ISO26262 part 10) because it takes into account their specific clauses, and they are gathered in our safety case. A component usage conceptual model (Fig 8) is defined for our ROS based architecture based on [31]. This means that each usage model is configured by three previously defined parameters: period, granularity and queue. Each configuration is related to a s pecific reliability based on the tests results (Table 1). We can therefore derive different reliabilities based on different configurations.

4.3.2 Measurable attributes

According to [31] we need to identify different profiles for the usage of our SEooC component. At least we need to identify the reliability behaviour for the ROS based architecture [43]. A quantitative technique for profiling the runtime behaviour is based on testing workloads [44]. We test our ROS based architecture by sending messages. In this sense stress testing [45] has been used in the past, and it is an essential activity in safety-critical software [46].

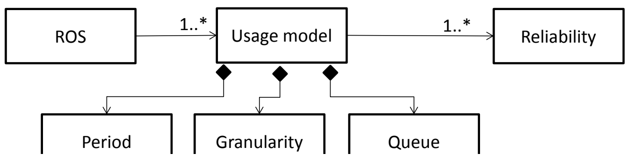


Fig 8: Component usage conceptual model

Software reliability models require failures intensity measurement to work [43] [47] [48] [49]. Therefore we need to define in our context what a failure is. ISO26262 part 1 [10]] defines fault as an abnormal condition that can cause an element (1.32) or an item (1.69) to fail. An error is a d iscrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition. Finally, a failure is a termination of the ability of an element (1.32) to perform a function as required.

In our definition, we are just considering as faults basically lost messages. A lost message means, for example, that the information sent to an actuator is not received, and therefore not processed. These faults can also be considered as relative errors in this context [28]. In consonance with some experiences reported in the literature using, among others, number of failures, probability and mean [29] [50] [51] [52], we are going to measure the following aspects for each experiment:

- Faults: number of undelivered messages or messages arrived after a cer tain latency threshold.
- Mean [29]: mean latency of all arrived messages.
- Median [29]: central latency of all arrived messages.
- Standard Deviation [51]: variation of arrived messages.
- Minimum: minimum latency value of all arrived messages.
- Maximum: maximum latency value of all arrived messages.
- Confidence Interval (CI) 95% (lower and upper): thresholds delimiting outliers [50] [51]. We only consider outliers latencies higher than Upper CI 95 %.
- Undelivered Density [52]: number of undelivered messages per total number of messages sent.
- Outlier Density [52]: number of outliers per total number of arrived messages.
- Reference Interval (RI) 99% (lower and upper): thresholds delimiting 99% of the arrived messages.

4.3.3 Black box testing performance

Following Voas' proposal, we aim at determining whether the component quality is high enough [25]. Fig 9 describes the steps carried out for testing our SEooC component. Tests are grounded on the analysis of SEooC component attributes: period, granularity and queue size. For each attribute value combination the platform is executed, recording the latency for each arrived message. As an implementation decision, non-arrived messages (undelivered) are marked as zero latency. Although it is not a valid latency value it is visually easy to spot.

One aspect to analyze is whether these data follow a normal distribution or an SRGM in order to evaluate the reliability of the experiments determined by these measures: mean, median, standard deviation, minimum, maximum, CI 95% (lower and upper), fault density and RI 99% (lower and upper).

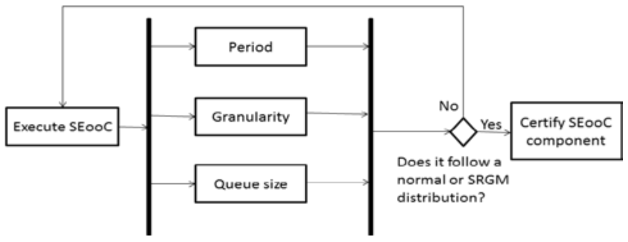


Fig 9: Black box testing process

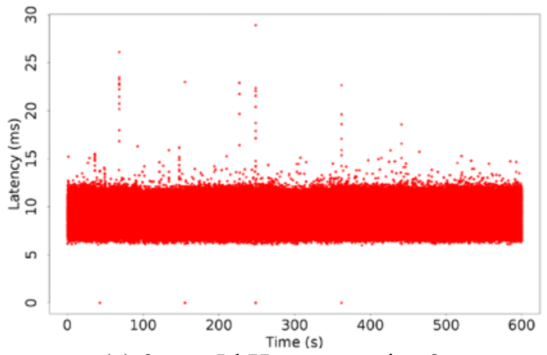
4.3.4 Results

We need to assess software operational quality [53] based on stressing our architecture. So, we identified 40 different configurations (5 x 2 x 4 = 40) for our SEooC component, being their input variables and values:

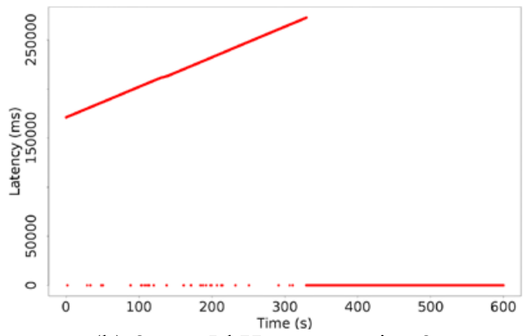
- **Period** values: 2, 3, 4, 5 and 8 ms.
- **Granularity** values: 5 and 10 kHz.
- **Queue size** values: 1, 2, 10 and 100.

Each experiment combination was executed 24 times during 10 minutes, therefore resulting altogether in 960 executions (see Annexes). These empirical cases provided an overview of the SEooC component behaviour under stress conditions.

In a preliminary data analysis we determine which experiments were meaningful. Fig 10 depicts the latency of two executions of out SEooC component. X axis represents the execution time in seconds (10 minutes altogether). Y axis represents the latency of each message over time in milliseconds. Undelivered messages are shown as having null latency. Fig 10 (a) shows a high density of messages delivered within the range from 6 to 12 ms latency. Four spots close to the X axis (zero latency value) correspond to undelivered messages. On the upper zone of the thick stripe, loose dots represent messages with a high-out of common latency. Fig 10 (b) depicts a completely different scenario. At the beginning, some messages are delivered with an extremely high latency (more than 150 seconds). Further to a given point (around 300 seconds execution time) messages no longer arrive, so that they all remain null (representing undelivered messages). Clearly, in this case the component cannot be considered reliable as messages mostly remain undelivered. The whole experiment set with a period of 2 ms behaves similarly. Furthermore, they are not comparable and we shall exclude them from the remaining of our study. This reduces our data set to 768 (24 executions of 4 x 2 x 4 = 32 experiments).



(a) 3 ms, 5 kHz, queue size 2



(b) 2 ms, 5 kHz, queue size 2

Fig 10: Data plots of two executions of the SEooC component

Similar figures to those provided in Fig 11 are obtained for the whole set of 768 valid executions and in general terms they all look like Fig 11 (a). We have visually compared the figures of the 24 executions under the same experimental conditions and they are consistent. All summarized information has been collected and compared in spread sheets with two different sorting views: (1) blocks of the 24 executions under each experimental condition and (2) blocks of all experiments under the same execution number. All this information can be consulted in the Annexes.

Another visual representation utilized in our analysis has been the density plot. Fig 11 shows an execution instance of our SEooC. In this picture we have integrated some of the extracted aforementioned measures with the values for this case.

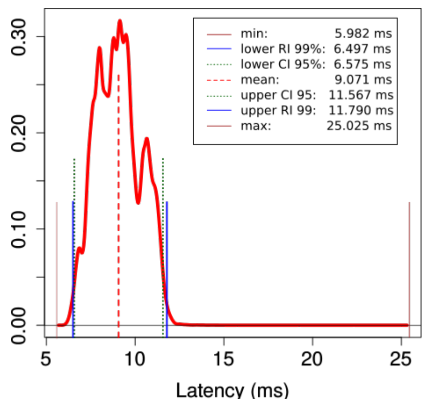


Fig 11: Density plot of execution 19, period 3ms, granularity 5 kHz, queue size 1

In order to formally analyze the experimental results we need to check if they follow a n ormal distribution to apply the Central Limit Theorem. When sample size is 8 t o 29 (in our case 24), we need to verify whether the Shapiro-Wilk and Kolmogorov-Smirnov normality test is fulfilled. Since the mean latency values do not violate the normal assumption, the Confidence Intervals (CI) 95% can be calculated as in Equation (1).

$$\text{Confidence Interval 95\%} = \mu \pm 2\sigma \qquad (1)$$

Table 1 synthesizes this information by combining the average of all 24 execution measures under each experimental condition.

4.4 Report & Certificate

This final activity for the certification process involves producing a report containing the details of the process, the safety case, the ROS architecture description, the usage model and the final results.

5 Discussions

ROS is a set of open-source software libraries. It is becoming popular in several domains but it is just in its infancy and its maturity can be debatable.

As demonstrated in this paper, it can be configured in different specific usage models so that each usage model may be related to a specific reliability level. In fact our tests reveal some relevant behaviour depending on these attributes: period, granularity and queue size.

Table 1: Summary of SEooC component executions

Period (ms)	Granularity (kHz)	Queue size	Messages	Minimum (ms)	Maximum (ms)	Std. Deviation (ms)	Mean latency (ms)	Lower CI 95 (ms)	Upper CI 95 (ms)	Undelivered msgs	‰Undelivered msgs	Outliers	% Outliers	Lower RI 99 (ms)	Upper RI 99 (ms)
3	5	1	200k	5.98	26.45	1.24	9.06	6.57	11.54	18k	90.27	2400	1.32	6.48	11.78
3	5	2	200k	5.99	25.91	1.30	9.22	6.61	11.82	15	0.08	2191	1.10	6.54	12.06
3	5	10	200k	5.99	30.38	1.31	9.24	6.63	11.85	24	0.12	2219	1.11	6.56	12.11
3	5	100	200k	5.99	28.89	1.30	9.23	6.62	11.83	23	0.12	2229	1.11	6.54	12.09
3	10	1	200k	5.96	26.79	1.30	9.03	6.43	11.62	4k	19.69	2035	1.04	6.44	11.79
3	10	2	200k	5.98	27.50	1.31	9.04	6.42	11.66	17	0.09	2258	1.13	6.44	11.88
3	10	10	200k	5.99	27.01	1.31	9.04	6.42	11.67	19	0.10	2194	1.10	6.44	11.88
3	10	100	200k	5.99	27.37	1.31	9.04	6.42	11.66	21	0.11	2232	1.12	6.44	11.88
4	5	1	150k	5.93	25.11	1.31	9.09	6.46	11.71	459	3.06	1489	1.00	6.36	11.98
4	5	2	150k	5.93	26.83	1.32	9.10	6.46	11.74	8	0.05	1512	1.01	6.37	12.04
4	5	10	150k	5.92	25.73	1.32	9.10	6.46	11.74	8	0.05	1490	0.99	6.36	12.03
4	5	100	150k	5.93	26.57	1.32	9.09	6.45	11.73	7	0.05	1476	0.98	6.36	12.01
4	10	1	150k	5.89	25.24	1.26	8.98	6.46	11.51	151	1.01	1787	1.19	6.37	11.72
4	10	2	150k	5.89	27.05	1.27	8.99	6.46	11.53	11	0.07	1787	1.19	6.38	11.74
4	10	10	150k	5.87	28.01	1.27	8.97	6.44	11.50	9	0.06	1847	1.23	6.37	11.73
4	10	100	150k	5.91	26.41	1.27	9.01	6.47	11.55	10	0.07	1717	1.14	6.40	11.75
5	5	1	120k	5.89	26.41	1.28	9.01	6.45	11.57	23	0.19	2631	2.19	6.34	12.02
5	5	2	120k	5.90	26.00	1.28	9.01	6.45	11.57	8	0.07	2609	2.17	6.33	12.02
5	5	10	120k	5.89	26.20	1.28	9.01	6.45	11.57	8	0.07	2638	2.20	6.34	12.01
5	5	100	120k	5.91	27.03	1.28	9.01	6.45	11.58	9	0.08	2608	2.17	6.33	12.02
5	10	1	120k	5.85	27.76	1.29	8.99	6.40	11.57	17	0.14	1493	1.24	6.37	11.78
5	10	2	120k	5.83	27.21	1.30	8.97	6.38	11.56	9	0.08	1525	1.27	6.35	11.76
5	10	10	120k	5.82	26.40	1.30	8.96	6.35	11.56	10	0.08	1611	1.34	6.34	11.76
5	10	100	120k	5.85	26.32	1.30	8.97	6.37	11.56	9	0.08	1566	1.31	6.35	11.77
8	5	1	75k	5.85	23.99	1.31	8.93	6.30	11.56	2	0.03	128	0.17	6.27	11.32
8	5	2	75k	5.84	23.49	1.31	8.92	6.30	11.55	2	0.03	133	0.18	6.27	11.31
8	5	10	75k	5.83	24.57	1.31	8.92	6.30	11.54	2	0.03	136	0.18	6.27	11.32
8	5	100	75k	5.86	24.45	1.31	8.93	6.30	11.55	1	0.01	128	0.17	6.27	11.32
8	10	1	75k	5.78	23.08	1.23	8.81	6.35	11.27	3	0.04	313	0.42	6.16	11.25
8	10	2	75k	5.80	24.42	1.23	8.81	6.35	11.26	2	0.03	302	0.40	6.15	11.25
8	10	10	75k	5.79	23.88	1.23	8.81	6.35	11.27	2	0.03	301	0.40	6.15	11.25
8	10	100	75k	5.79	24.25	1.23	8.80	6.34	11.26	2	0.03	298	0.40	6.15	11.25

Our ROS based architecture behaviour is characterized in terms of timing and, therefore, our research question 1 (RQ1: What is the timing behaviour of a ROS based architecture?) is answered. A different sensor, controller and actuator configuration can generate other values and therefore require other reliabilities. However, it is useful to understand how a single ROS configuration can interact with other components, and to know its acceptable thresholds. The usage model represents different configurations and reliabilities, and it is the designer's responsibility to decide upon which reliability he/she wants to run.

Our safety case provides a set of argumentations and asserted inferences for supporting ISO26262 and the SEooC concept. All these assumptions and asserted evidences supporting goals argue that a specific instantiation of a ROS based architecture is acceptably safe and compliant to some ISO26262 clauses. So, our second research question (RQ2: What are the safety aspects that should be taken into account during the safety case d efinition?) is therefore accomplished.

A certification process is a double edged sword [54], and a b alance of activities should be defined. The proposed certification process deals with specific ISO26262 practices related to SEooC. A certification process should clarify what evidences are the most relevant but also what are the main ISO26262 aspects involved for a SEooC certification. This article provides an example on the arguments, assertions and evidences used in this context. This leads us to conclude that our third research question (RQ3: What aspects should be highlighted during a certification process for a ROS based architecture?) is reached. However, these arguments and assertions and evidences can vary depending on each situation and context. If this ROS component were integrated in a more complex scenario, additional arguments and evidences should be defined.

Our ROS based architecture is tested on a basic platform which is not a complex situation where several ROS and other components are interacting. More studies are required on complex infrastructures and mechanisms. For example, the current bus communicator is a Universal Serial Bus (USB), and traditionally communications in automotive domain use the so called CAN bus [55]. However it can be used with USB for connecting other automotive components. In addition, further research should be devoted to security issues which are not covered in CAN bus systems representing a main weakness.

The presented approach helps us to identify what use of a ROS component can be defined by the designer. In fact, it is under the designer's responsibility to accept or to decline a sp ecific configuration according to one specific application requirements. Our ROS characterization provides an overview of the acceptable values considering Confidence Intervals (95%) and Reference Intervals (99%). During a certification process these values are taken as reference values. These aspects are used during ISO26262 SEooC component definition.

6 Conclusions and future work

Software is playing a key role in automotive industry, and ROS has its place for specific activities. This paper presents a characterization of a ROS based architecture with a deep analysis of its behaviour according to three parameters: period, granularity and queue size. Certifying third party components in the automotive industry should consider the ISO26262 SEooC concept definition. Our ROS is defined as a SEooC component, and it is used on a vehicle for testing purposes. The presented approach defines a cer tification process compliant to ISO26262 and to the SEooC concept. A set of reference values is defined to be used during assessment activities. These values represent a usage model, and a specific reliability is related to each usage model.

This certification approach relies on the use of safety cases for representing primary documentation. This documentation must include reference values for a SEooC component, and these values are provided in this paper. Our safety case provides a s et of argumentations and asserted inferences for supporting ISO26262 and the SEooC concept. All these assumptions argue that a specific instantiation of a ROS based architecture is acceptably safe and it is compliant to some ISO26262 clauses. At the end we provide a ROS based architecture analysis and its compliance to ISO26262 SEooC.

As future work, one relevant aspect is to model ROS behaviour as an SRGM, but further analysis should be devoted to define this SRGM model. We are currently working on a preliminary proposal in this sense but some formalism is required to present this model. In addition, we have identified behavioural improvements modifying the ROS kernel. In this sense there are some interesting initiatives such as Linux for automotive[4], and how ROS can be smoothly integrated with this operating system.

Annexes

A more detailed description of the experimental environment and the individual execution results are available in data and graphical modes at the following annexes web address:

http://lsi.vc.ehu.es/CPS-data

Acknowledgments

This work was supported by the Basque Government Project CPS4PSS Etortek14/10.

References

[1] Jo K, Sunwoo M. Generation of a Precise Roadway Map for Autonomous Cars. IEEE Trans Intell Transp Syst 2014;15:925–37. doi:10.1109/TITS.2013.2291395.

[2] Kerr J, Nickels K. Robot Operating Systems: Bridging the gap between human and robot, 44th IEEE SouthE Symp System Theory; 2012, 99–104. doi:10.1109/SSST.2012.6195127.

[3] Zubrycki I, Granosik G. Test setup for multi-finger gripper control based on Robot Operating System (ROS), 9th IEEE Intl Workshop Robot Motion and Control; 2013, p. 135–40. doi:10.1109/RoMoCo.2013.6614598.

[4] https://www.automotivelinux.org/

[4] Sen Z, Lei S, Zhongliang C, Lishuang Z, Jingtai L. A ROS-based smooth motion planning scheme for a home service robot, 34th IEEE Chinese Control Conf; 2015, p. 5119–24. doi:10.1109/ChiCC.2015.7260438.

[5] Noh S, Park B, An K, Koo Y, Han W. Co-Pilot Agent for Vehicle/Driver Cooperative and Autonomous Driving. ETRI J 2015;37:1032–43. doi:10.4218/etrij.15.0114.0095.

[6] Hawkins R, Habli I, Kelly T, McDermid J. Assurance cases and prescriptive software safety certification: A comparative study. Safety Science 2013; 59:55–71. doi:10.1016/j.ssci.2013.04.007.

[7] Gallina B. A Model-Driven Safety Certification Method for Process Compliance, IEEE Intl Symp Soft Reliability Eng Workshops; 2014, p. 204–9. doi:10.1109/ISSREW.2014.30.

[8] Larrucea X, Combelles A, Favaro J. Safety-Critical Software [Guest editors' introduction]. IEEE Software 2013; 30:25–7. doi:10.1109/MS.2013.55.

[9] Areias C, Cunha JC, Iacono D, Rossi F. Towards Certification of Automotive Software, IEEE Intl Symp Software Reliability Engin; 2014, p. 491–6. doi:10.1109/ISSREW.2014.54.

[10] International Standard Organisation. Road vehicles – Functional safety; ISO 26262, 2011.

[11] Adedjouma M, Hu H. Process Model Tailoring and Assessment for Automotive Certification Objectives, IEEE; 2014, p. 503 –8. doi:10.1109/ISSREW.2014.23.

[12] Mader R, Armengaud E, Grießnig G, Kreiner C, Steger C, Weiß R. OASIS: An automotive analysis and safety engineering instrument. Reliability Eng & Syst Safety 2013;120:150–62. doi:10.1016/j.ress.2013.06.045.

[13] OpenCert: Evolutionary Assurance and Certification for Safety-Critical Systems n.d. https://www.polarsys.org/introducing-opencert-evolutionary-assurance-and-certification-safety-critical-systems (Accessed March 16, 2017).

[14] Hawkins R, Habli I, Kelly T, McDermid J. Assurance cases and prescriptive software safety certification: A comparative study. Safety Science 2013; 59:55–71. doi:10.1016/j.ssci.2013.04.007.

[15] Dale C, Anderson T, editors. Advances in Systems Safety. London: Springer London; 2011.

[16] Steele P. Certification-based development of critical systems, IEEE; 2012, p. 1 575–8. doi:10.1109/ICSE.2012.6227033.

[17] Ayoub A, Kim B, Lee I, Sokolsky O. A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments. In: Ortmeier F, Daniel P, editors. Comput. Saf. Reliab. Secur., vol. 7612, Berlin, Heidelberg: Springer Berlin Heidelberg; 2012, p. 305–16.

[18] Dardar R, Gallina B, Johnsen A, Lundqvist K, Nyberg M. Industrial Experiences of Building a Safety Case in Compliance with ISO 26262, IEEE; 2012, p. 349 –54. doi:10.1109/ISSREW.2012.86.

[19] Wassyng A, Maibaum T, Lawford M, Bherer H. Software Certification: Is There a C ase against Safety Cases? In: Calinescu R, Jackson E, editors. Found. Comput. Softw. Model. Dev. Verification Adapt. Syst., vol. 6662, B erlin, Heidelberg: Springer Berlin Heidelberg; 2011, p. 206–27.

[20] Barry MR. CertWare: A workbench for safety case production and analysis, IEEE; 2011, p. 1–10. doi:10.1109/AERO.2011.5747648.

[21] Hernandez C, Abella J. Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems. IEEE Trans Comput-Aided Des Integr Circuits Syst 2015;34:1718–29. doi:10.1109/TCAD.2015.2434958.

[22] Lyu MR. Software Reliability Engineering: A Roadmap, IEEE; 2007, p. 153 –70. doi:10.1109/FOSE.2007.24.

[23] Hernandez C, Abella J. Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems. IEEE Trans Comput-Aided Des Integr Circuits Syst 2015;34:1718–29. doi:10.1109/TCAD.2015.2434958.

[24] Morris J, Lee G, Parker K, Bundell GA, Chiou Peng Lam. Software component certification. Computer 2001; 34:30–6. doi:10.1109/2.947086.

[25] Voas JM. Certifying off-the-shelf software components. Computer 1998; 31:53–9. doi:10.1109/2.683008.

[26] Musa JD, Everett WW. Software-reliability engineering: technology for the 1990s. IEEE Software 1990;7:36–43. doi:10.1109/52.60588.

[27] Musa JD. A theory of software reliability and its application. IEEE Trans Softw Eng 1975;SE-1:312–27. doi:10.1109/TSE.1975.6312856.

[28] Software Reliability. Reliab. Saf. Eng., vol. 0, London: Springer London; 2010, p. 193–228.

[29] Jung H-J, Yang H-S. Software Reliability Measurement Use Software Reliability Growth Model in Testing. In: Gervasi O, Gavrilova ML, Kumar V, Laganà A, Lee HP, Mun Y, et al., editors. Comput. Sci. Its Appl. – ICCSA

2005, vol. 3482, Berlin, Heidelberg: Springer Berlin Heidelberg; 2005, p. 739–47.

[30] Panesar-Walawege RK, Sabetzadeh M, Briand L. Using Model-Driven Engineering for Managing Safety Evidence: Challenges, Vision and Experience, IEEE; 2011, p. 7–12. doi:10.1109/WoSoCER.2011.8.

[31] Wohlin C, Regnell B. Reliability certification of software components, IEEE Comput. Soc; 1998, p. 56–65. doi:10.1109/ICSR.1998.685730.

[32] Bruyninckx H. Robotics Software: The Future Should Be Open [Position]. IEEE Robot Autom Mag 2008;15:9–11. doi:10.1109/M-RA.2008.915411.

[33] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, et al. ROS: an open-source Robot Operating System. ICRA Workshop Open Source Softw 2009;3:5.

[34] Staranowicz A, Mariottini GL. A survey and comparison of commercial and open-source robotic simulator software, ACM Press; 2011, p. 1. doi:10.1145/2141622.2141689.

[35] Noh S, Han W-Y. Collision avoidance in on-road environment for autonomous driving, IEEE; 2014, p. 884–9. doi:10.1109/ICCAS.2014.6987906.

[36] Nair S, de la Vara JL, Sabetzadeh M, Briand L. Classification, Structuring, and Assessment of Evidence for Safety - A Systematic Literature Review, IEEE; 2013, p. 94–103. doi:10.1109/ICST.2013.30.

[37] Ayoub A, Kim B, Lee I, Sokolsky O. A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments. In: Ortmeier F, Daniel P, editors. Comput. Saf. Reliab. Secur., vol. 7612, Berlin, Heidelberg: Springer Berlin Heidelberg; 2012, p. 305–16.

[38] Linling S, Wenjin Z, Kelly T. Do safety cases have a role in aircraft certification? Procedia Eng 2011;17:358–68. doi:10.1016/j.proeng.2011.10.041.

[39] Dodd I, Habli I. Safety certification of airborne software: An empirical study. Reliab Eng Syst Saf 2012;98:7–23. doi:10.1016/j.ress.2011.09.007.

[40] Fachet R. Re-use of software components in the IEC-61508 certification process. vol. 2004, IEE; 2004, p. 8–8. doi:10.1049/ic:20040532.

[41] Zeng F, Lu M, Zhong D. Software Safety Certification Framework Based on Safety Case, IEEE; 2012, p. 566–9. doi:10.1109/CSSS.2012.147.

[42] Hawkins R, Kelly T, Knight J, Graydon P. A New Approach to creating Clear Safety Arguments. In: Dale C, Anderson T, editors. Adv. Syst. Saf., London: Springer London; 2011, p. 3–23.

[43] Musa JD, Iannino A, Okumoto K. Software reliability: measurement, prediction, application. New York: McGraw-Hill; 1987.

[44] Sârbu C, Johansson A, Suri N, Nagappan N. Profiling the operational behavior of OS device drivers. Empir Softw Eng 2010;15:380–422. doi:10.1007/s10664-009-9122-z.

[45] Jiang B, Chen P, Chan WK, Zhang X. To What Extent is Stress Testing of Android TV Applications Automated in Industrial Environments? IEEE Trans Reliab 2015:1–17. doi:10.1109/TR.2015.2481601.

[46] Baker R, Habli I. An Empirical Evaluation of Mutation Testing for Improving the Test Quality of Safety-Critical Software. IEEE Trans Softw Eng 2013;39:787–805. doi:10.1109/TSE.2012.56.

[47] Jelinska Z, Moranda PB. Software reliability research. Stat. Comput. Perform. Eval., n.d., p. 465–84.

[48] Goel AL, Okumoto K. Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. IEEE Trans Reliab 1979;R-28:206–11. doi:10.1109/TR.1979.5220566.

[49] Davidsson M, Jiang Zheng, Nagappan N, Williams L, Vouk M. GERT: An Empirical Reliability Estimation and Testing Feedback Tool, IEEE; 2004, p. 269–80. doi:10.1109/ISSRE.2004.21.

[50] Xi J. Outlier Detection Algorithms in Data Mining, IEEE; 2008, p. 94–7. doi:10.1109/IITA.2008.26.

[51] Josephs HJ. The fixing of confidence limits to measurements. J Inst Electr Eng - Part II Power Eng 1945;92:194–206. doi:10.1049/ji-2.1945.0049.

[52] Hatton L. Reexamining the fault density component size connection. IEEE Software 1997;14:89–97. doi:10.1109/52.582978.

[53] Hamlet D. Theory of Software Testing With Persistent State. IEEE Transactions Reliability 2015; 64:1098–115. doi:10.1109/TR.2015.2436443.

[54] Matsubara T. Process certification: a double-edged sword. IEEE Softw 2000;17:104–5. doi:10.1109/52.895176.

[55] Davis RI, Burns A, Bril RJ, Lukkien JJ. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 2007; 35:239–72. doi:10.1007/s11241-007-9012-7.