

Utilizing UML and patterns for safety critical systems

Kai T. Hansen and Ingolf Gullesen

ABB Corporate Research Center Norway
Bergerveien 12; P.O.Box 90;

N-1361 Billingstad

Norway

Tel.: +47 66 843060 Fax: +47 66 843541

Email: Kai.Hansen@no.abb.com, ingolfg@hasle.no

Abstract.

This article discusses methods of object-oriented analysis and design in UML suitable for the specific needs developing safety critical software systems, and to which degree safety can be related to components. Different patterns, typical for safety related software, are mapped into UML models. The safety aspects are characterized by that the focus is not only on the fulfillment of functional requirements for the process but also on fulfillment of patterns such as diversity, diagnostics, safety layers and on development quality. These safety patterns can be viewed as a feature provided by the system when implementing a functional requirement and not as a primary functionality. This justifies the argumentation that a *Safety Quality of Service* should be employed and modeled aligned with the proposed standard for modeling resources in UML.

Introduction

In safety critical systems there are special requirements in order to obtain sufficiently reliability. Capturing requirements, analyze them, design and test the system systematically is demanding when one has to obtain complete complex system where the probability that an error leads to a dangerous situation may be up to order 10^{-9} per hour for control in process industry ref. [1]. The standard IEC 61508 ref. [1] will be the main safety standard referred to in this article. Douglass [2] has discussed some patterns and UML for safety-related systems. There is some discussion of requirement capturing and verification using UML for safety critical systems in the literature. In ref. [2] Carpenter addresses the question of requirement verification by focusing on how UML use cases can be used to specify requirements and to verify them. Carpenter is addressing the question on verification of requirements and how to trace these through a project. This is however not unique, only more important for safety critical systems. There exist some software tools intended to help tracking requirements through analysis and design of safety critical systems, some discussed in [4]. High quality is also required by safety system but is not unique either. The uniqueness of a safety critical system is that the design usually includes hardware redundancy, some type of software redundancy, software fault tolerance and substantial self-supervision diagnostic software. The system must be analyzed by one or more acknowledged hazard analysis methods to verify that the redundancy and diagnostic is sufficient. One method for obtaining some of the safety critical functionality is a monitor system as discussed in ref. [5]. There are an increasing number of computer and software systems that is of high-dependability and safety-critical type [6] and since UML is becoming the main design language it is important that UML provide the safety-critical software developers with well developed methods. There is progress in the effort to standardize both real-time and scheduling and also fault-tolerance in UML, ref. [7]. The hardware part of a system is usually well modeled by stochastic processes and it is well known that these can be quantified. The focus here is therefore on the software part where less is known.

Safety is a system concept and lifetime concept, and it is a huge simplification to assume it can be fully modeled as low-level components. However, to some degree it can be done this way and a final system safety analysis will rely mostly on safety aspects in the components.

There is a slight distinction between safety systems that has a safe state, e.g. chemical industry where the process can be stopped or a train that can slow down, and safety systems with no safe state, e.g. commercial airplanes. We have here the main focus on systems with a safe state.

Diversity of software

How safe a system is, is depending on how reliable the functionality is that ensures that the system is not entering a dangerous state. When we discuss a computerized control of a physical system, this translates to the reliability that the electronics and computer do not set output values that turns the system under control into a risk situation where life, environment, or equipment could be damaged.

One important way to increase reliability is redundancy. This is the traditional way to increase reliability of hardware. Some type of functional redundancy is also often used for software.

The requirement for a safety system will often include diversity functionality. This can be in the form of a diverse code execution, or in form of handling a COTS (Commercial Of The Shelf) SOUP (Software Of Uncertain Pedigree) problem with a protection function solution as sketched in the fault trees in Figure 1 (a) and (b).

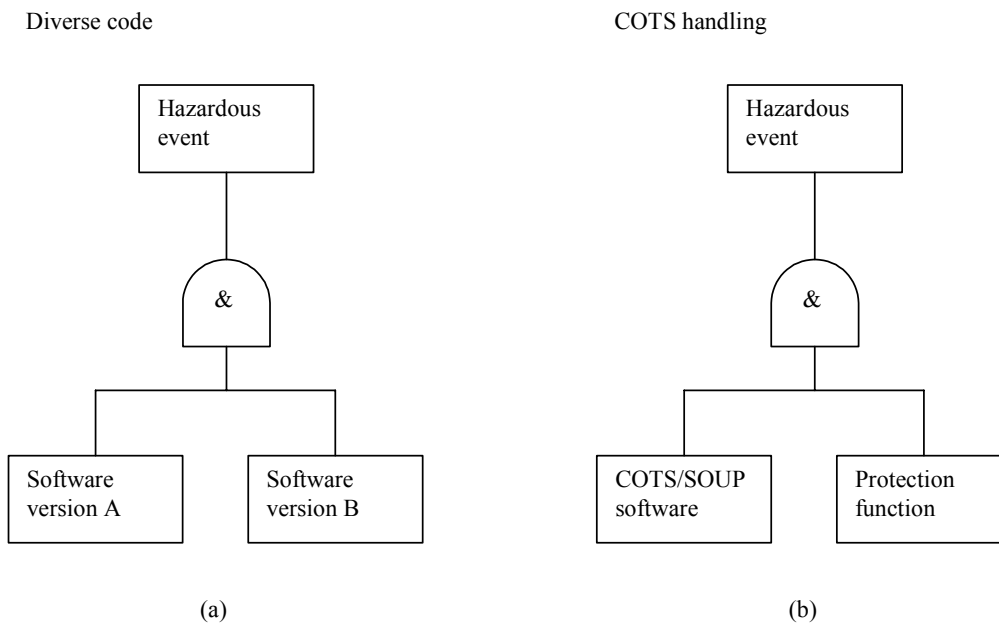


Figure 1 Requirement of diverse structures in safety critical systems, either as (a) full redundant system, or (b) as a protection function to a functional code, possibly of COTS type.

The main requirement for the system is in fact the functionality one software version in Figure 1 (a) is providing, or the COTS functionality in Figure 1 (b). The redundancy shown here is a mean to provide a sufficient level of safety or reliability.

Safety Quality of Service

The ideal way to provide the functional requirements with the proper safety level would be to introduce a Quality of Service (QoS) for safety. This might be done in the same manner as in ref. [8] where properties related to time are modeled. In Figure 2 there is sketched such a resource providing a QoS for a functionality requiring safety.

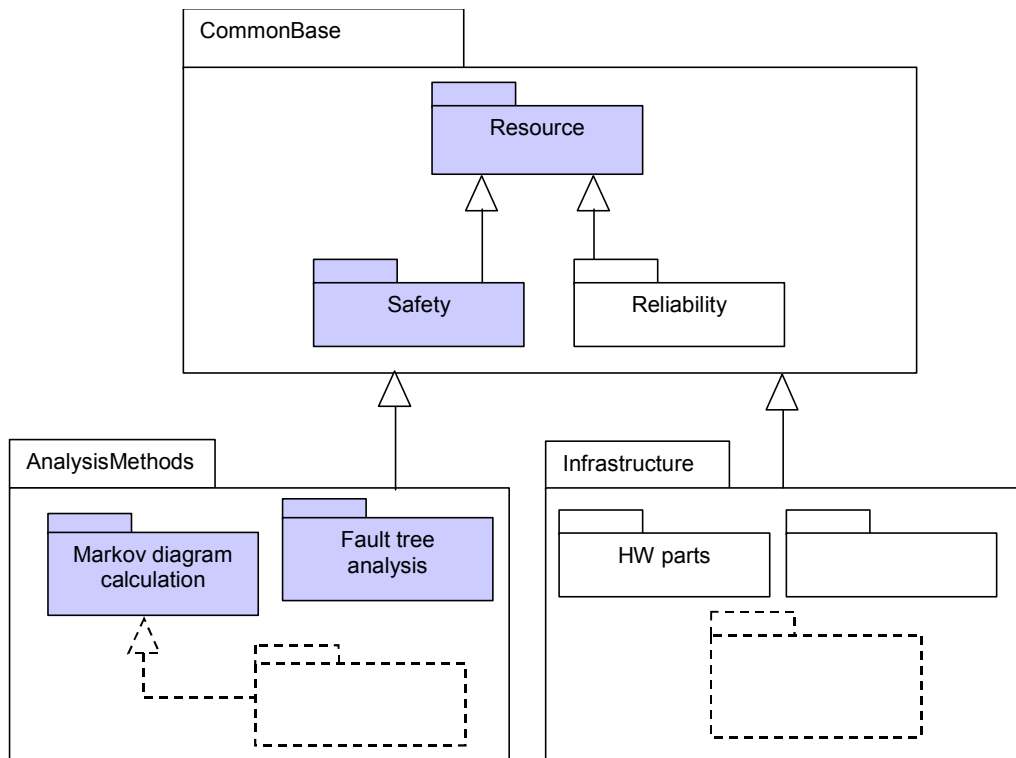


Figure 2 Resources in Safety systems

The resource modeling in Figure 2 proposes that a resource may have a Safety QoS as well as a Reliability QoS. This should then be a QoS in the same way as Time and Concurrency can be modeled. It also proposes that these qualities can further be analyzed with suitable methods and the infrastructure described. The quantitative analysis methods available for safety are Markov theory and simulation techniques in addition to simpler fault tree analysis. This involves both hardware and software and the hardware problem is the simplest as this can usually be modeled as a stochastic process.

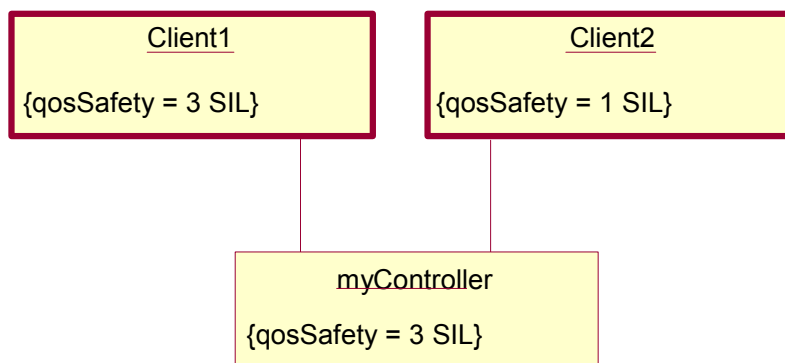


Figure 3 Two-layer QoS specification

Assuming the successful modeling of a safety QoS this can be employed as sketched in Figure 3. Here the two clients are requiring a specific safety level. Here the safety level is specified as a SIL number. SIL is the Safety Integrity Level specified in the standard IEC 61508 on functional safety for computer systems, ref. [1].

A successful application of this philosophy requires a proper modeling of a resource. It is also not clear that the software modeling to obtain high safety level can be modeled in this way. A protection function as in Figure 1 (b) may have a complex interaction with the functional code. An important part of acknowledging a high SIL is that specific software diagnostics or patterns have been used and that the development quality is at a given level. These more qualitative measures can be used as a QoS for a component even without having a quantitative analysis method.

Dual channel modeling in UML

A simple full diverse functional example is shown in Figure 4. Here the use case simply states the requirement of a given functionality and that this from input to an output shall be independent and diverse. The actors are here the physical field values that sensors gives and actuators receive.

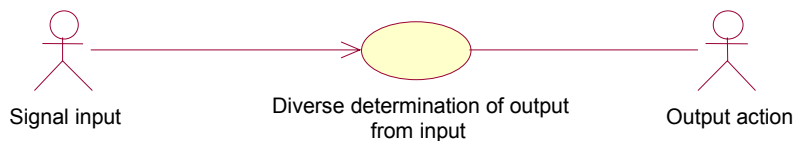


Figure 4 A use case description of a diverse functionality

A main flow of event is drawn as a sequence diagram for the use case of Figure 4 in Figure 5. Here the value from the input is taken to a common object and distributed to the objects representing software version A and software version B. These objects will provide all application specific logic to the signals and determine an output signal. It is necessary to pipe this signal through a common object (voter) to determine the signal to use. The common front-end and back-end objects are relatively simple and may be implemented in hardware, but cannot be avoided in the functional description of the diverse system.

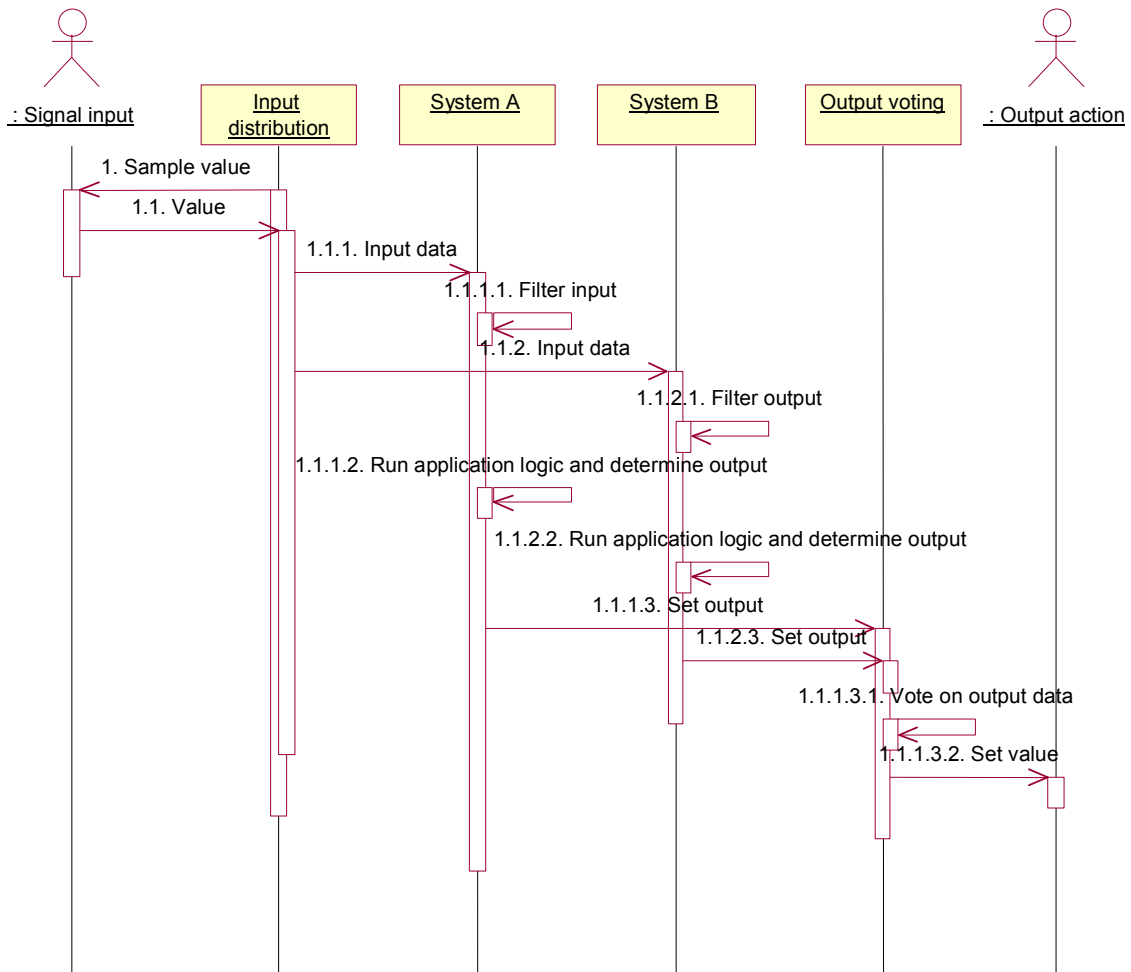


Figure 5 Sequence diagram

The system in Figure 4 could be viewed as an UML system in a use-case view. Then we can define UML subsystems such that these subsystems add up to the system. This is illustrated in Figure 6.

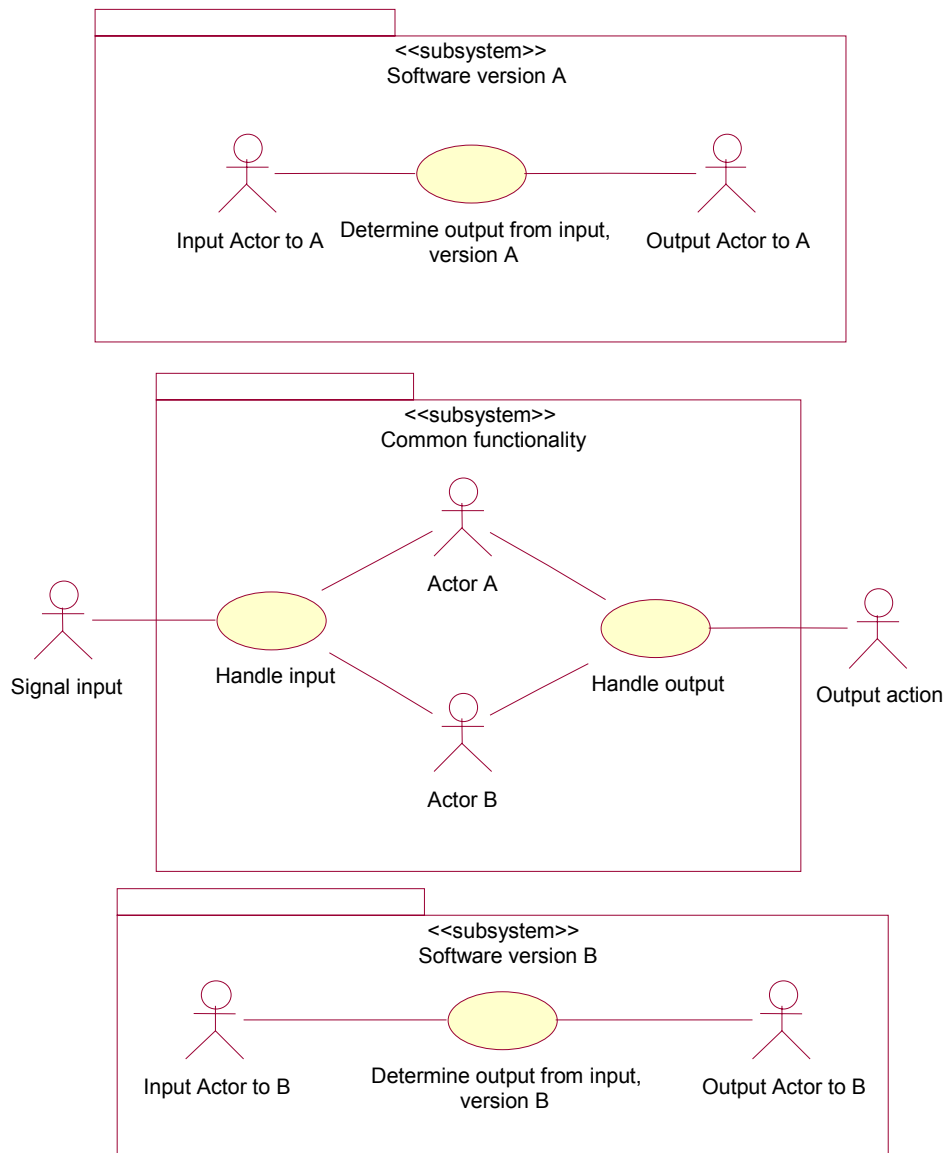


Figure 6 Splitting diverse functionality in diverse UML subsystems

The subsystems A and B are the two channels providing diversity in this design. The subsystem “Common functionality” provides the necessary glue to merge the A and B subsystems into the diverse system. As we see from the sequence diagram in Figure 5, in this example the Common functionality only consist of a simple distribution of input data and a collection and voting of output data. This is the traditional full dual channel approach. The functional requirement of determining output from input according to a specified logic is now almost completely inside subsystem A and repeated in subsystem B. This gives a relatively simple way to relate a functional requirement to the two subsystems A and B.

The three subsystems can be shown to add up to the full system. We identify the object “Input distribution” in Figure 5 (which is an object in the “Common functionality” subsystem) with the role of actor “Input Actor to A” in subsystem “Software version A”. This object also plays the role of “Input Actor to B” in

subsystem “Software version B”. The objects in Figure 5 can in this way all be identified with actors in Figure 6.

The next step in the analysis phase is to develop further subsystem A and subsystem B. Two independent teams, if necessary, can do this, since the interface between the subsystems is defined at a requirement level. Based on the requirement of how output shall be determined from input, the two teams can make distinct sequence diagrams or other UML diagrams to determine objects and algorithms for the solution of the problem. In the design phase both teams can design distinct classes, components and interactions but the Common functionality subsystem has to be designed in cooperation with the teams to provide a correct interface to each subsystem. In many systems the classes in the Common functionality subsystem will be implemented by hardware components, but this do not make any difference for the analysis and design work.

The functionality of this example is to determine output signals from input signals according to some logic. The use case in Figure 4 is an expression of this requirement but here the demand that it shall be done in a diverse way is explicit. A better requirement would be a statement that the output shall be determined from the input according to some logic with a Quality of Service of Safety Integrity Level 3 as in Figure 7. It would then in the analysis phase be a focus on the actual functionality. The determination that a SIL 3 should in this case be obtained by a diverse realization of the functionality would be postponed to a slightly later time in the analyzes- and design phase. This would be according to the ideas expressed in Figure 2 and Figure 3. The safety QoS would then naturally be associated with a component cover the whole use case functionality of Figure 4. This component would then have the functionality of determining output from input and in addition have a SIL 3 QoS. As we discuss below, this QoS will only be in respect to the diversity aspect, while other aspects such as development quality also must be fulfilled and a system analysis must be done. Hardware redundancy may also come in addition.

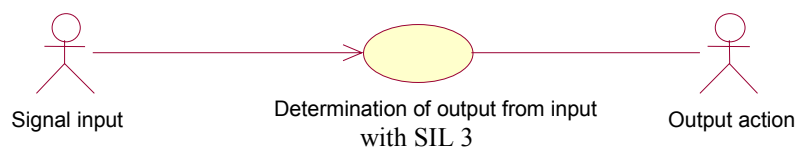


Figure 7 A use case description of SIL 3 functionality

This way to split a system into subsystems not only in the design model but also in the use-case model may appear unusual, but is necessary if the requirement shall be fulfilled that two completely independent developments shall be made. In this case only the requirements shall be common and the split in subsystems must therefore be done at use case model level.

The subsystem in Figure 6 has only one (possibly complex) requirement that is the main logic. A requirement that often is wanted and gives much more problems is the use case where an operator is allowed to reconfigure the system while executing the main logic. It may require some type of synchronization between the two subsystems and this is not good as a common cause fault can be introduced. This can give more problems than the main requirement and must be handled with care.

Partial diversity and diagnostic

The approach of obtaining high safety integrity level by implementing a complete diverse structure is a traditional approach. It has advantages such as giving less risk of dangerous systematic faults, it is easy to explain and defend and it can easily be extended to more channels giving further reduction of safety risk. However, it has some drawbacks and may not always be the preferred method to obtain high safety Quality of Service.

One major problem with the method is that it is very expensive in developing time and human resources. As all real-life projects has a limited budget and time span, the cost of full diverse channels will eventually take recourse away from tasks such as requirement handling, quality of analysis and design, and from testing and verification. A dual channel design does not help at all against the problem of wrong requirements. If the logic specified to be implemented in subsystem A and subsystem B in Figure 6 has a fault or is lacking description of some subtle exception handling, then both systems may fail the same way in a critical situation for the system under control and may cause an accident. In modern control systems there are often so many advanced features interacting with a control loop and diverse implementation of all these bells and whistles are beyond the budget of any project and not necessary for the safety of the primary control. Having a system with major diagnostic functionality to ensure the correctness of the functionality combined with some diversity of implementation is often a much solution than a full diverse channel system.

The UML modeling of a one-channel system with diagnostic is more difficult to split into subsystems at an early stage of the analysis work, such as we described for a simple two channel diverse solution.

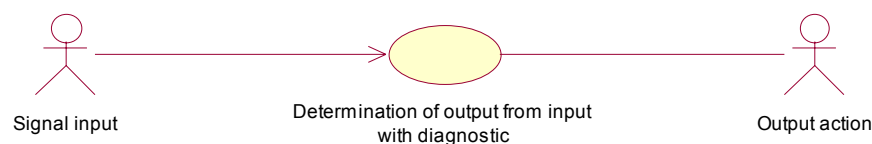


Figure 8 Use case description for system with diagnostic

One method covering parts of the needs is to define a shadow object for every relevant object in the functional design. The shadow object will do diagnostic or diverse execution for the functional object. This is illustrated in Figure 9.

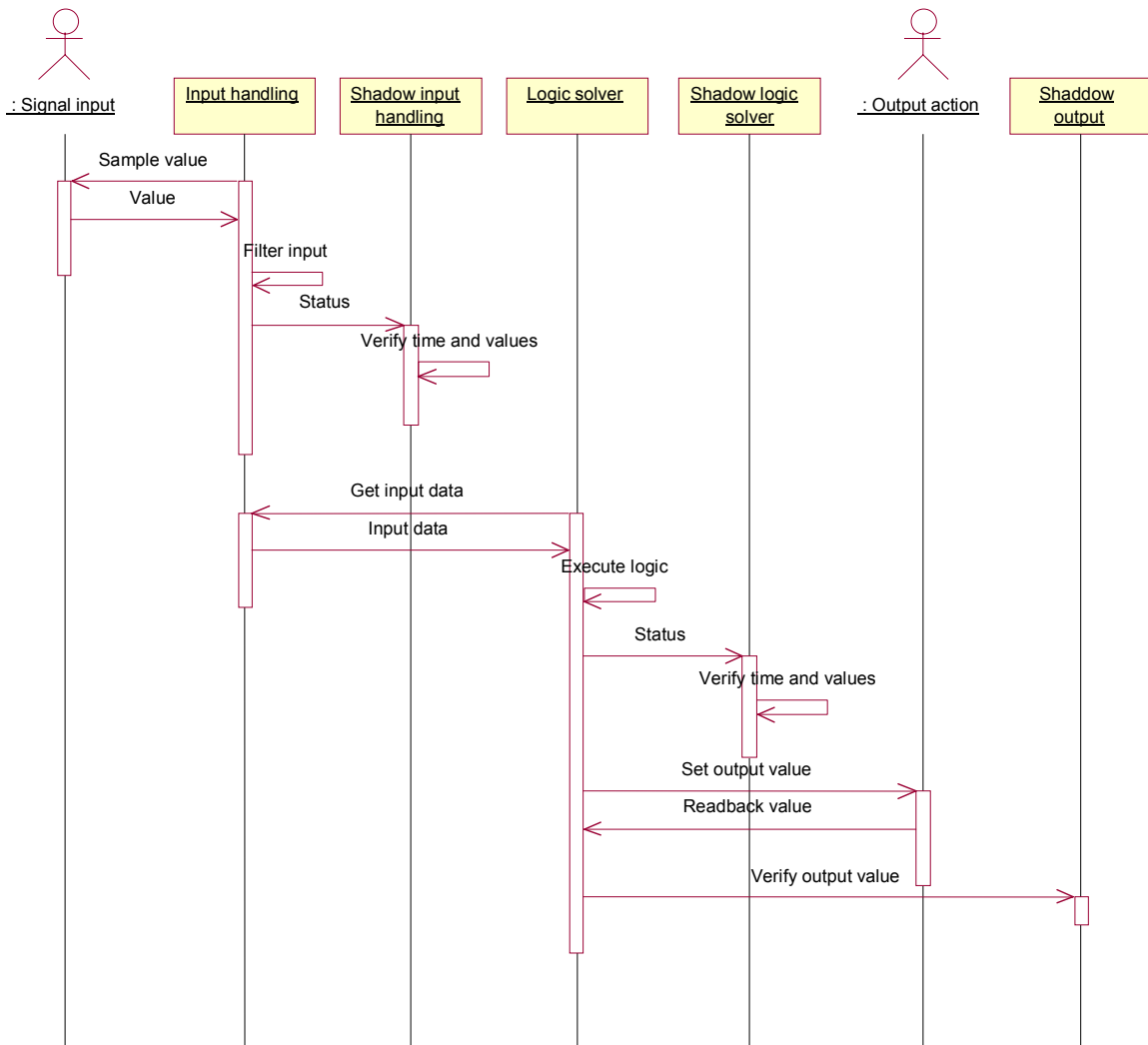


Figure 9 Sequence diagram with shadow objects

It is possible to split the system into subsystems such that all shadow objects are part of a shadow subsystem and the remainder in a different subsystem. This will give subsystems with large number of use cases and large amount of messaging between the subsystems. Figure 10 shows a simple example with three shadow safety objects. Such a split may not be advantageous in all cases and the diagnostic may be modeled as a part of the main functionality. However unless split into a subsystem it may be more difficult to utilize QoS ideas as in Figure 3.

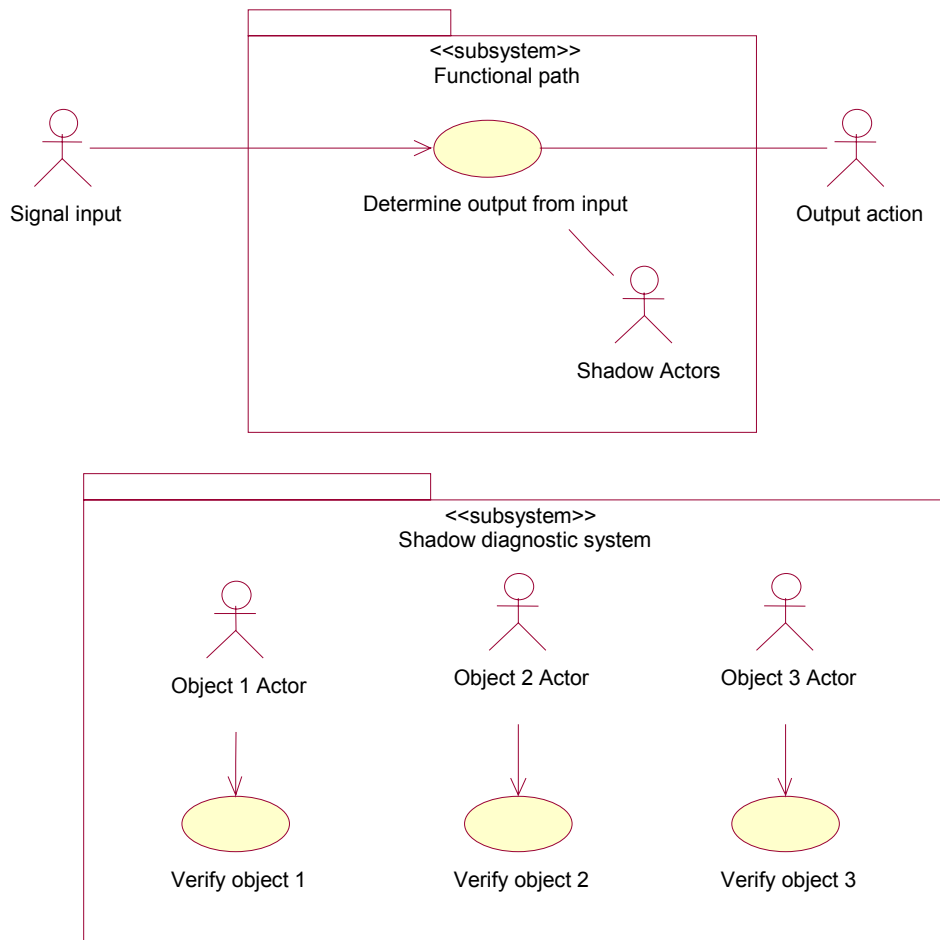


Figure 10 Splitting of shadow safety diagnostic in a subsystem

A safety shadow object may contain both verification parts and diverse execution parts. A dual channel may be view upon as a very special shadow object where there is a shadow path of the same complexity as the original path. Normally will a shadow object have a much more limited functionality.

A scheduling object, e.g. in an operating system, may be supervised by a shadow timing verification object which compares scheduling times with a different list of acceptable timings for each task in the system. It may base the comparison on a time source originated from a different physical crystal and check both that tasks are not delayed but also that it is not too fast in order to obtain a high diagnostic coverage (e.g. ref. [1] part 2 tables A10 and A12 and in part 7 section A9.2).

In this way we can declare a scheduler to be an object that has a QoS of SIL 2 or SIL 3. That is; the complete system will have this safety level with respect to dangerous faults resulting from a scheduling fault that this object is responsible for. It is necessary to always be aware that the safety integrity level is a system feature and only obtained when combining the components to a system.

A different example could be a control logic object which executes the “business logic” of the system that may have a shadow object do some type of supervision of

the logic. This may range from monitoring that the RAM storing the code do not change, monitor in some way parts of the logic, or have a complete diverse implementation of the logic.

Some objects may have a shadow object with sanity check pattern. This verifies that the calculations give a reasonable answer in some sense.

A shadow object of this type may even involve a human evaluation and possible reaction. The time scale of reaction will be less predictable and fast when a human interaction is involved, but may sometimes be necessary. The post-reaction features such as event list presentations to an operator is typically not a system feature increasing the safety level of the design, even though it is necessary for the functionality.

The safety QoS is then associated with the main object or component that has implemented the required functionality, and the shadow object provides the SIL.

Safe communication and gray channel pattern

There are different types of safe communication needs for safety critical systems. In particular there are differences between real-time and not real time needs, e.g. download of new code versus getting input data from a remote sensor. It is also difference between systems with a safe state such as control in dangerous chemical industry where it is possible to stop a process versus a system that requires constant control as in flight management of airplanes. In high demanding bus systems used in flight-by-wire and drive-by-wire bus systems there is hard real-time demands and often a time triggered philosophy is used as TTP, SAFEbus and others as discussed in ref. [10]. These have strong requirements for both real-time and for being able to functional even if some nodes fail in any possible way. In applications where it is necessary that information is correctly delivered but not as tough real-time requirements one can accept to close the system if a node fails or is too late in response. What is common for all the safe communication links is that they can be build by having a safety layer in the protocol stack in some way. Also for a high demanding bus like TTP the physical bus can be a COTS solution like the CAN bus or a RS485 bus with its physical lines and driver chips. However the TTP protocol takes decisions on when and what each not shall send on the physical layer. Systems made for safe transportation of configuration data before starting etc. may use much more of an existing COTS system but will put a safety layer on the top in order to verify that the content has not changed, it has reached the intended addressee, it is within specified time, etc. The part of the communication channel below the safety layer is called the gray channel and in an object-oriented language, the safety layer will become a object or class. This type of pattern is a safety communication pattern in UML.

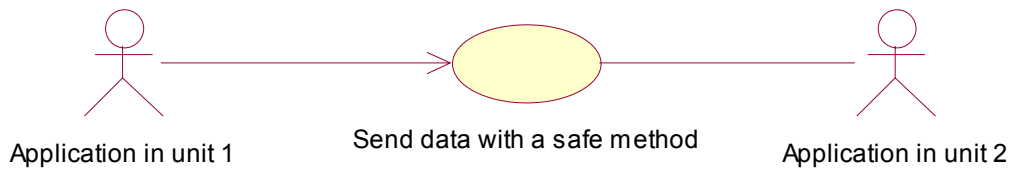


Figure 11 A use case description of safe communication

If the sequence involves objects that have the pattern of a communication protocol a safety layer can be used (e.g. [11]). This can be modeled as an additional objects inserted between the objects viewed as the application and the object identified as the top layer in the protocol. This then defines the protocol as a gray channel and is sketched in Figure 12 and Figure 13. This makes it possible to define gray channel collection of objects, which are not safety related, as any fault in these will be found by the safety layer objects. The gray channel can be hidden behind an interface and may be considered to be a subsystem as in Figure 14. The safety layer object must be able to both discover any fault in transmission of frames as well as lack of communication. The subsystem can here also be only made in the design model and not in the use case model as in Figure 14.

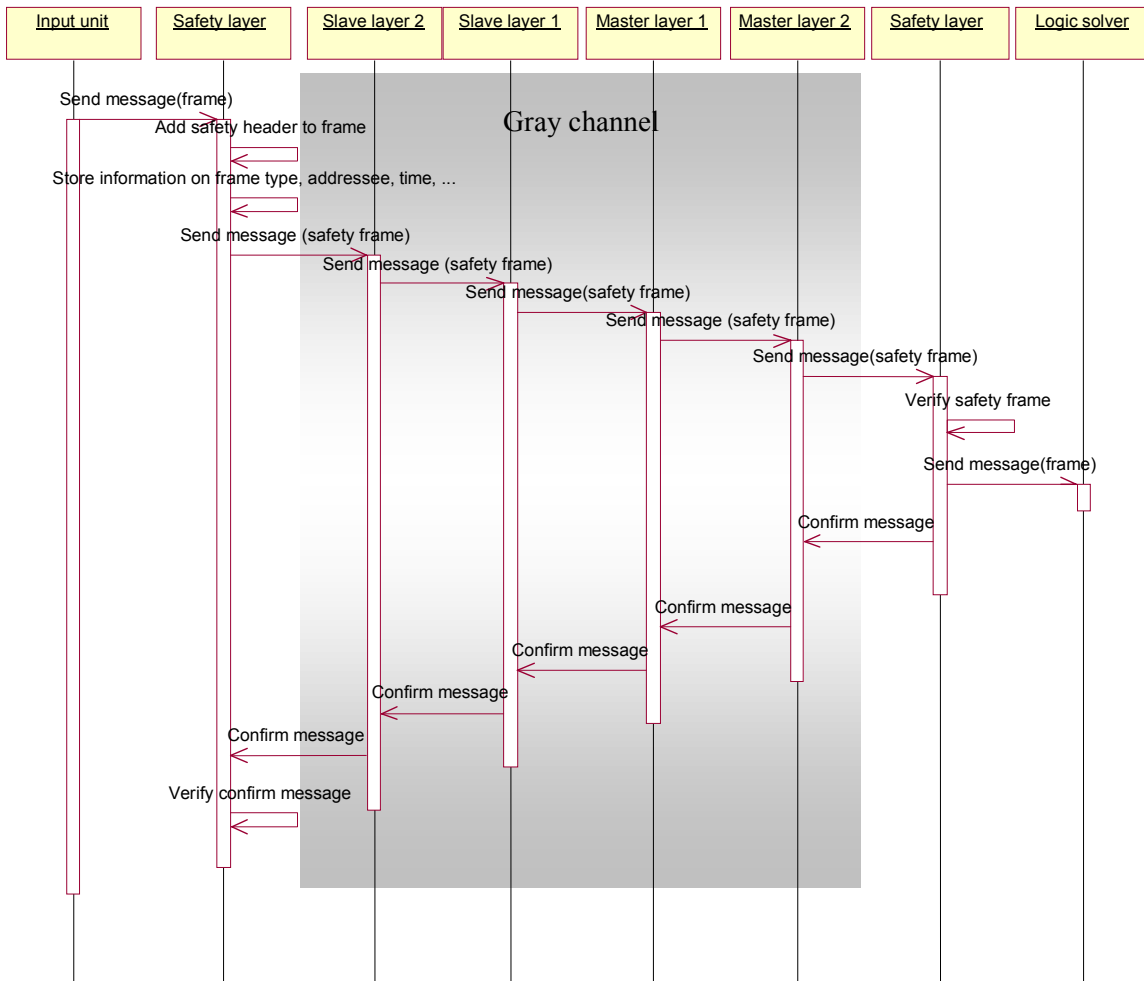


Figure 12 Fragment of sequence diagram with safety layer objects and gray communication channel

The gray channel pattern is not limited to a standard protocol stack. It can be used to cover any communication method. A communication involving open Internet can be made safe with sufficiently advanced safety layer, but hard real-time will of course not be obtained in such a solution.

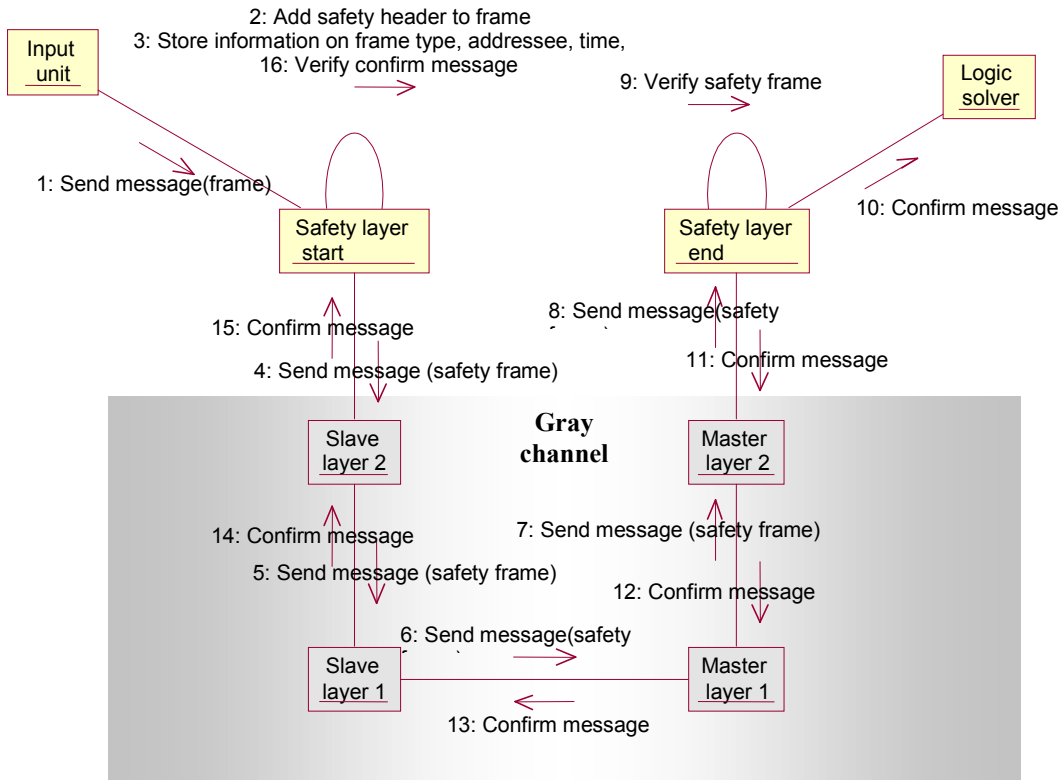


Figure 13 Safe communication gray channel as collaboration diagram

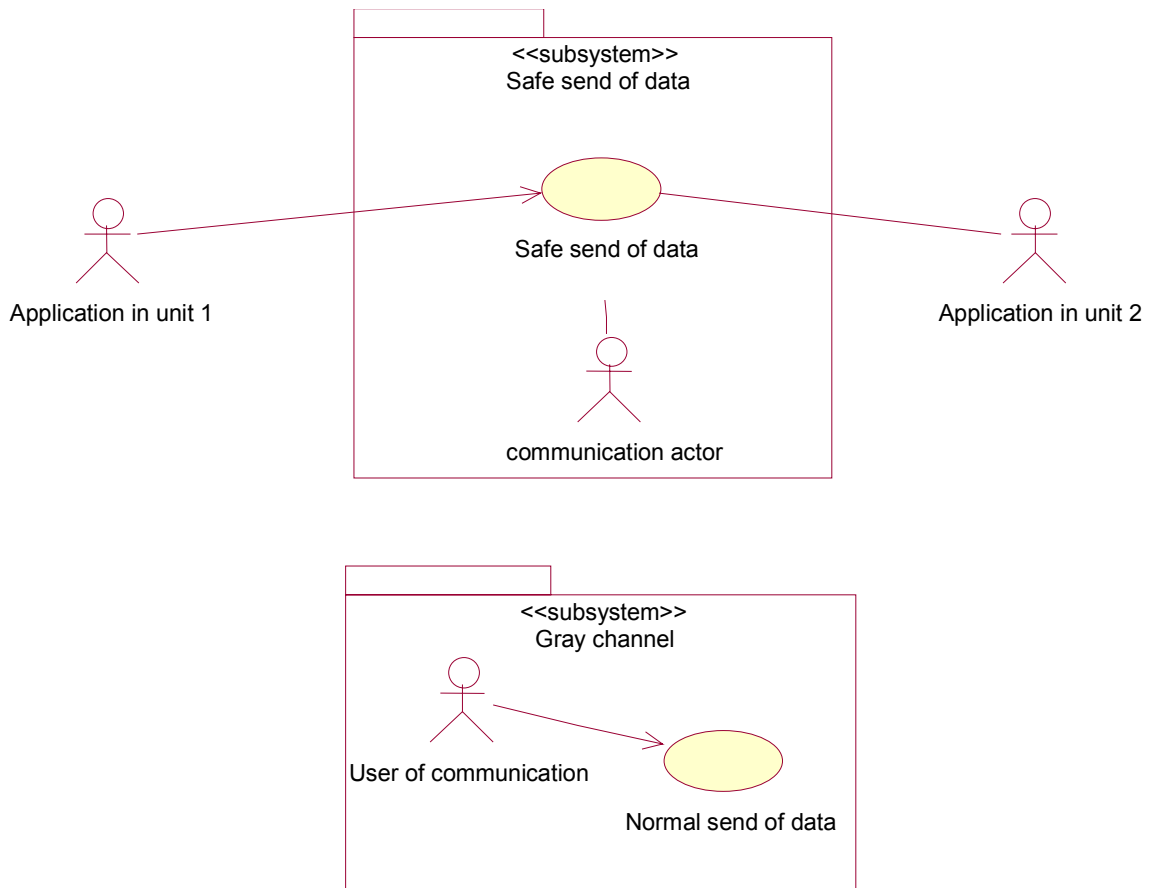


Figure 14 Safe communication subsystem organization

Testing strategy

The testing of a system build with UML is discussed in e.g. [12] and [13]. There is also a proposal for UML testing profiles available [14]. Testing can be focused around scenarios using sequence or collaboration diagrams as a test is done as a sequence. Partial testing of a system under development is also possible by replacing some objects by simple stubbing. In [14] it is tried to formalize the modeling of this.

For safety critical systems it is an additional interest to do testing that validates the diagnostic functionality. A natural way to provide this is by employing a fault injection technique. This can be done in different ways in UML [15]. A test strategy to verify the safety aspects of the system and software design must include a systematic fault injection test configuration suitable for the safety pattern chosen.

The dual channel pattern with the splitting in subsystems, Figure 6, requires a test configuration with test cases where fault injections are done either in subsystem A or in subsystem B, but not the same fault in both simultaneously. The stimulus to the test is legal input data to the system, e.g. typical sensor data, which also will be used for normal functional requirement verification tests. The verdict is done on the specified test behavior, which could be that all fault injections shall be discovered in the “Handle output” use case in the “common functionality” subsystem, Figure 6. This will normally be a functionality like the “vote on output data” in Figure 5. It is

however a slightly too simple strategy if the goal of the test architecture is to validate the safety level of the system. It would be sufficient to only discover those faults that lead to dangerous situations for the system under control. This can be done if sufficiently is known about the whole system architecture, but with the lack of this it may be a first strategy to do the verdict depending on if the fault is discovered in the “common functionality” subsystem. The strategy chosen will be project depended, but requiring that all injected fault shall be discovered may lead to a far to expensive and complex system. The test architecture of the dual channel pattern also illustrates the well-known weakness of this safety pattern: a common mode failure in the design will not be discovered. A common mode fault in both subsystem A and subsystem B corresponds to a test case where the same fault is injected in both subsystems and is therefore not found in the “common functionality” subsystem. The probability of a common mode fault can be reduced by designing A and B very differently, but can never be completely eliminated because e.g. a fault in the requirements will give the same fault in the implementation.

Figure 15 shows an example on the scenario expected with a fault injection in subsystem B of the dual channel pattern. In this scenario the correct handling of the fault injection shall be to set the output to a predefined safe state.

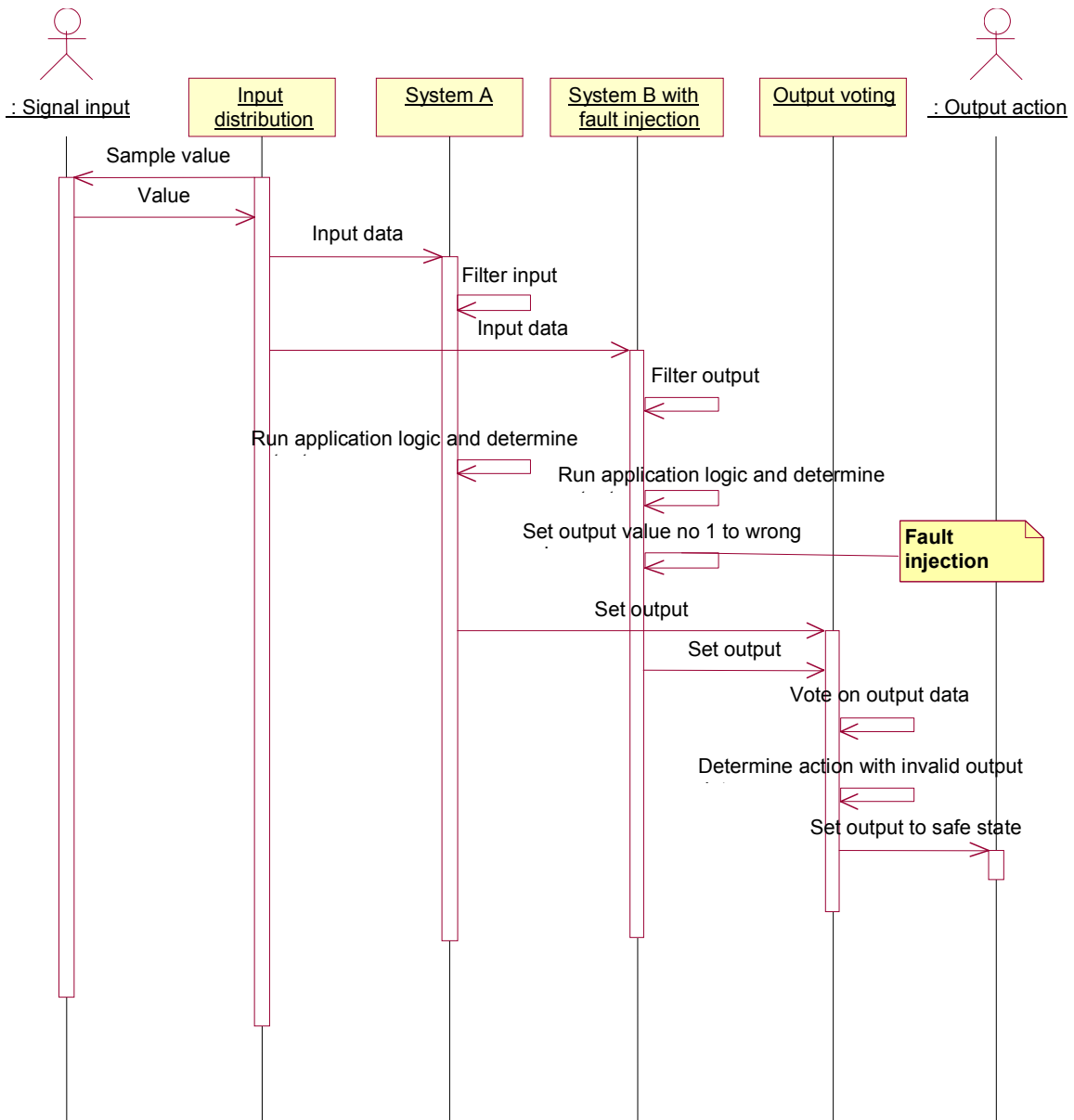


Figure 15 A sequence diagram for a dual channel test case.

For the other two safety patterns discussed, similar fault injection test cases are defined. For the shadow safety diagnostic pattern in Figure 10 fault injection is done in the “Functional path” subsystem and the verdict is depending on that the “Shadow diagnostic system” subsystem is taking appropriate action resulting e.g. in setting output to safe state.

For the Safe communication subsystem pattern in Figure 14 the fault injection is done in the “Gray channel” subsystem and the verdict is based on that the “Safe send of data” subsystem will find the fault in its safety layer object and handle this in the pre-described way, e.g. setting output to safe state.

The test cases should be run in the analysis phase, the design phase and during design in addition to a formal testing at the finished system. The running of the test will then range from a pure analysis “gedanken” experiment to executing the complete code on the target system.

Hazard analysis

It is always necessary to do some type of hazard analysis of a safety related system and there exist several methods for this, e.g. ref. [1] part 7 section B6 and C6.2

There is some relation between the testing strategy discussed above and a hazard analysis of the system. A full handling of all wrong behavior from all software objects is beyond any possible efforts in a complex system. A more sensible way to find a large part of all software faults leading to a dangerous system is to employ e.g. software HAZOP analysis [16]. From the HAZOP analysis it may come out some messages between objects that have some possibility to appear and could lead to a dangerous situation. These identifications would lead to a redesign of the system and one would probably employ one of the safety patterns added to the original functional design. These would also be candidates for the test cases discussed above in order to verify that the system design is handling properly this potential risk situation.

Quality development

A safety standard also set requirements to the development process. Depending on the SIL level there are requirements to the formality of the design (ref. [1] part 3 tables A1, A2 and A4 and part 7 sections B2, B3.2 and C2). An object designed in UML with a proper methodology will normally satisfy this requirement up to SIL 3. Often SIL 4 will require formal methods and be difficult to obtain with normal methods using UML. A safety QoS regarding the development process can be obtained for a component or object, by connecting it to the documentation of the development process used.

There will also be requirements on the quality of implementation, which programming language to use, etc. (ref. [1] part 3 table A3 and part 7 section C4). A safety QoS regarding the implementation is obtained by connecting it with a description of the implementation methodology used.

Conclusion

We have discussed of some different aspects when utilizing UML for a safety related system. We believe that a system with a safety integrity level of 1, 2 or 3 can be obtained by combining a complete system analysis with using UML components with a safety QoS inherent in the components and classes. It is therefore useful to associate safety QoS to components even though the safety of a system is, and must be, based in an analysis of the complete system.

References

- [1] IEC 61508 - *Functional safety of electrical/electronic/programmable electronic safety-related systems*. 1998. (<http://www.iec.ch/>)
- [2] B. P. Douglass. *Doing Hard Time: Using Object Oriented Programming and Software Patterns in Real Time Applications*, Addison-Wesley Publishing, 1999.
- [3] Carpenter, P. B.. *Verification of Requirements for Safety-Critical Software*. SIGAda'99 10/99 Redondo Beach, CA, USA.
- [4] Faller, R. SAFECOMP 2001.

- [5] Peters, D.K.; Parnas, D.L. *Requirements-based monitors for real-time systems*. Software Engineering, IEEE Transactions on, Volume: 28 Issue: 2, Feb. 2002, On page(s): 146 – 158.
- [6] Kopetz, H. Software engineering for real-time: a roadmap. *The Future of Software Engineering 2000, Proceedings 22nd International Conference on Software Engineering*, ACM Press 2000 pp 201-211.
- [7] <http://www.omg.org>
- [8] Response of the OMG's Request for proposal on Schedulability, Performance, and Time
(http://www.omg.org/techprocess/meetings/schedule/UML_Profile_for_Scheduling_FTF.html).
- [9] OPC specification, see <http://www.opcfoundation.org/>
- [10] John Rushby, *Bus Architectures for Safety-Critical Embedded Systems*, EMSOFT 2001, LNCS 2211, pp. 306–323, 2001.
- [11] DeviceNet Safety protocol. <http://www.can-cia.de/devicenet/dnetsafety.pdf>
- [12] Lionel Briand and Yvan Labiche, *A UML-Based Approach to System Testing*. LNCS 2185, p. 194.
- [13] Marc Lettrari and Jochen Klose, *Scenario-Based Monitoring and Testing of Real-Time UML Models*. LNCS 2185, p. 317.
- [14] Response of the OMG's Request for proposal on UML Testing profile
(http://www.omg.org/techprocess/meetings/schedule/UML_Testing_Profile_RFP.html)
- [15] Alaa Ibrahim., HH Ammar, S. Yacoub, *A Fault Model for Fault Injection analysis of UML Dynamic Specifications* ISSRE 2001
- [16] Felix Redmill, Morris Chudleigh and James Catmur, *System Safety: HAZOP and Software HAZOP*. John Wiley & Sons, Chichester 1999.