IAEA Technical Meeting

On Licensing Digital Instrumentation and Control Systems and Equipment in Nuclear Power Plants

Espoo, Helsinki, Finland, 22-25 November 2005

Towards a Deductive Approach for the Safety Justification of Computer Based Systems

P.-J. Courtois[1]
AVNuclear[2], Brussels, Belgium

Not only should the system be dependable,
it should also be possible to demonstrate to the regulator
that it is dependable
(IAEA Safety Guide NSG1.1, §3.19)

Abstract

The objective of this on-going research work is to find ways to make the demonstration of the safety of a computer based system as deductive as possible; that is, to put in place concepts, principles and methods to structure and rationalize this demonstration.

1. Introduction

When a programmable system with safety demands is on the verge of being introduced into service, those responsible are almost always haunted by the same doubts: is the system and its software sufficiently reliable? What can happen if, despite all precautions taken, it fails? Shall it be fail-safe…? Not to raise these questions would of course be foolish. And yet looking for answers raises more issues: Of all the possible evidence and arguments, which one should we most rely upon? How can we decide whether there is enough evidence to justify the release of the software system? How should evidence be presented to independent assessment bodies or authorities?

These questions are not new, even in nuclear engineering, see among others [2],[8]. And yet, past lessons learnt and other good reasons (see for instance [12]) strongly advise us not to rely exclusively in those matters on common sense and engineering judgement. Safety is indeed an elusive system property that may thwart experienced expertises, especially when correctness of design is at stake. There is a need for a rational, transparent, objective, and reproducible decision procedure to guarantee that

[1] Email: courtois@info.ucl.ac.be
[2] Rue Walcourt, 148, B-1070 Brussels

the necessary level of dependability is attained. The work reported here fits in with this general objective.

By necessity, the hardware and software of a complex software-based system is designed as a highly modular and hierarchically organised structure. The specification of the design and implementation is achieved through different refinement levels: specifications of the interface with the outside world are reified into the system architecture, the hardware and software design, implementation, and operation modes. A method which exploits the properties of this hierarchic reification structure is proposed here to construct safety arguments for a computer-based system.

Recommendations are given as to how to organize the justification of initial dependability claims into a multi-level structure of claims, evidence and arguments. This multi-level structure rests in particular on the concepts of *levels of evidence*, *claim* (as distinct from requirement), *claim conjunction, inference, expansion and delegation*. Levels of evidence are shown to correspond to levels of causality. To each level are also associated the corresponding system functions and undesired events susceptible to cause these functions to fail.

An as yet unexplored albeit essential aspect of dependability cases is emphasized: the essential roles played by models. The availability of reified models is critical in the demonstration of dependability, in two different ways: they are indispensable to define safety and also to deliver proofs.

This approach may complement assessment practices more oriented on rule-, design principle- or standard- compliance. Benefits that can be brought to current practices are discussed. In no way the paper intends to make believe that safety justification is simple. On the contrary, the expansion of apparently simple claims is shown to be surprisingly complex. Mastering this complexity is the concern.

Over the years, our work has been partly supported by the European Union, within the context of the R&D projects PDCS (Predictably Dependable Computing Systems), DeVa (Design for Validation), and CEMSIS (Cost Effective Modernization of Systems Important to Safety) [11].

2. How to start a safety justification? The primary dependability claims

We first have to define precisely what is meant by safety in the precise context in which the *computer system* is to be used. Curiously enough, nuclear guidance on safety critical computer systems in general devotes little or no attention to these first steps.

By *computer system*, or more simply *system*, we mean here a computer with its processors, memories and communication hardware, and with its system and application software. When we need to refer to the Instrumentation and Control system in which the *computer system* is embedded and operates, we shall talk of the *computer-based system*.

2.1 Initial dependability requirements for the computer based system

We have to start with a carefully validated definition and formulation of a complete and coherent set of *dependability requirements* for the computer-based system. Without such validated requirements, there would be no definition of the safety that must be claimed for the use of the computer system. This set of initial requirements should therefore receive the prior attention and approval of all parties involved, plant and safety engineers, and the regulator.

A couple of examples may help to keep things concrete.

In Tihange 1 (Belgium) nuclear plant, the SIP (**P**rocess **I**nstrumentation **S**ystem) is the input processing level of the Protection Logic System (PLS) of the reactor. Signal acquisition, validation, voting and threshold comparison are among its main functions; it consists of four independent and data acquisition and processing channels.

The SIP renovation and replacement by a digital system was completed in 2004. At the start of the project, some of the main dependability properties required for the new digital instrumentation were agreed upon:

Functionality:	**The SIP I/O functional relations must conform to the original logic.**
Reliability claim:	**The SIP reliability and availability must remain at least as good as the original one.**
CMF:	**The SIP software common mode failure (CMF) level must not be superior to the CMF level of the original SIP hardware being replaced.**
Single Failure:	**No single failure should cause the loss of a protection function.**
Auto-detection**:**	**The SIP must detect its own failures (in particular, the coverages achieved by auto-tests and periodic tests must be shown complementary).**
Fail-safeness**:**	**The SIP must be fail-safe.**

In some cases, only a couple of dependability properties need to be required. For example, in the project CEMSIS [11], the dependability expected from a rotational nuclear material-handling machine is:

> "No rotation" requirement**: Rotation of the carousel never occurs while material is being transferred**
> Reliability requirement*:* **The probability of failure per demand should be less than 10^{-4} per handling operation**

These initial requirements concern the functionality of the system or the properties of its behaviour. *Ultimately, however, all requirements have to be mapped and translated onto specifications of the computer system design, implementation, maintenance and/or operation control.*

2.2 Formulation of primary dependability claims

The next step is the derivation from the initial dependability requirements of what we call the *primary dependability claims* (or in short *primary claims*) of the computer system. *Primary claims* are still independent of the actual implementation; they basically claim (i) that the system *specifications* are valid and (ii) that the system *behaviour*, as specified, is dependable, taking only into account the environment, replacement or upgrade constraints.

Primary dependability claims are therefore of two sorts:

(i) A *functional primary claim* claims *quality properties (correctness, completeness, coherency…)* of the *specifications of the functions* expected from the *computer system* in a given context of constraints and anticipated hazards. The most simple and generic form of a functional primary claim is given in Figure 1.

(ii) A *non-functional primary claim* claims *properties of the implementation, maintenance or use* of the computer system. At this early stage, nonfunctional dependability properties of the implementation such as fail-safeness, reliability or availability cannot yet be mapped onto specific aspects of the lower design and implementation levels. So, non-functional primary claims take simple forms, as for example in Figure 2.

.

specification
{under {environment constraints[i]}**do**
if condition [i] **then** action [k];
if undesired events[i] **then enter** safe state S[j]**}**
is valid.

Comments:
valid means correct, consistent, complete with respect to a dependability requirement
constraints are those imposed by the environment (e.g. by physical constraints, existing systems and communication links, replacement constraints…)
conditions are the operational conditions that the safety system has to deal with.
actions are safety feature actuation, alarm generation, control…
undesired events are anticipated events taking place at the plant-system interface, also at lower levels of the design and implementation, but not necessarily yet identifiable at this stage.
safe state S is a system output state, which is dictated by the environment and defined here.

Figure 1

2.3 Claims, Meta- and Extra-Requirements

Primary claims, like system and design requirements, concern properties of the computer system specification or behaviour. There are, however, important differences between the notions of *claim* and *requirement*.

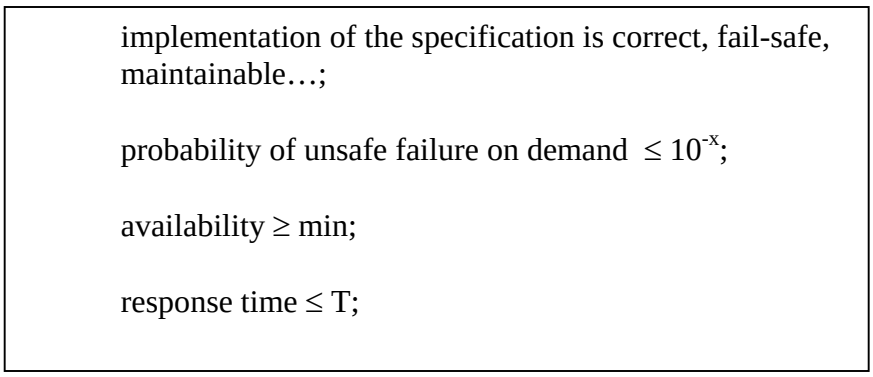

Figure 2

System architects and designers refer to system requirements and their specifications in the course of design. They normally do not talk about claims at this stage. One usually starts to make claims on a computer system when an application has to be made for licensing the system for a given usage, or when dealing with regulators, safety authorities or their technical support organisations, or when submitting the system to independent assessment.

A requirement is essentially a statement specifying what the computer system is originally designed to do, while a claim is a statement addressing the appropriateness(i.e. properties) of these specifications and of the system behaviour for a given usage.

More precisely, a claim is either a kind of meta-requirement or an extra-requirement.

A claim is a kind of *meta-requirement* when it is a statement addressing a *property* that a (set of) computer system requirement specifications should have: e.g. validity, consistency, completeness, correct implementation, or maintainability in operation.

A claim is a kind of *extra-requirement* when it is a statement addressing needs that were not explicitly part of the original computer system requirement specifications; such an extra-requirement can be a functional or a non-functional claim, depending on whether it claims the existence of extra functionality or property. For instance, one may have to claim that a monitoring system is fit for operations in post accidental conditions. It must then be claimed and shown to satisfy the design criteria - e.g. the single failure criterion and a maximum reaction time - required for these operations, although some of the system components may not have been originally specified and designed for this purpose.

Claims and requirements may coincide in contents in the ideal situation when the safety justification is part of a development project and progresses along with the specifications and the design of the system.

Components of the shelf (COTS) and other pre-existing components used in systems important to safety are an important case where claims usually do not coincide with the original system specifications.

A more subtle difference is that design requirements emphasize system functions while dependability claims also cover undesired events like threats or hazards. For example, a functional specification for a monitoring system may be “to display the pressurizer coolant temperature and level values”, while a functional dependability claim may have to exclude silent failure modes for the same system and demand that

"pressurizer coolant temperature and level values be displayed *if and only if validated measurements are available*". A computer system may satisfy its functional requirement specifications, and yet not necessarily satisfy a functional dependability claim.

2.4 Coherency and Completeness

It is essential that the set of primary claims be sound, consistent (no two claims contradicting each other) and complete (no essential claim missing). This is one of the first great difficulties of the demonstration of safety.

The primary claims are derived from the initial dependability requirements. The discussion of the soundness, completeness and consistency of these initial requirements is out of the scope of this paper. These initial requirements must result from prior analysis; otherwise, there would be no definition of the dependability expected from the I&C computer-based system to start with, and thus no agreement on what should be justified.

As for the primary claims, justifying their consistency and soundness may be relatively easy since the initial dependability requirements can be used as a reference to compare with. Justifying the completeness of the conditions, events, and actions that must be taken into consideration can be much harder. There are at least two pragmatic ways to help with this problem.

First, an important point to keep in mind is that completeness is a model property; that is, a property that can only be demonstrated in relation to a well-defined pre-existing model. For the *primary claims*, a model of the interface between the computer system and its environment is needed to precisely describe the environment items, conditions, actions, constraints, events and properties of interest. Completeness of the set of primary claims can then be shown by comparison with sets of elements defined by this model. The justification of completeness is transposed to the justification of a model; but the latter can be easier, more precise and less prone to errors. The need for precise models in safety justification is discussed in section 5.

Another important principle to remember is what R. P. Feynman coined a "principle of scientific thought that corresponds to a kind of utter honesty" [7]. Not only what supports the validity of the primary claims but also everything that may cast doubt on their completeness should be explicitly reported. Every potential event, condition or dependability issue that might have been eliminated from the *primary claims* by some experiment or some other argument should be documented so that independent reviewers can be aware of any aspect left aside and why.

3. How can we justify the primary dependability claims?

3.1. Hierarchical Structured Justification

A first key is to remember that any computer system hardware and software is highly modular and hierarchically organised. Its design and implementation specification is achieved through different refinement stages. Specifications of the interface with the

outside world are reified into specifications of the system architecture, design, implementation, and operation modes.

The second key is to realize that, similarly to this reification process, any primary claim is a property of the system specification or behaviour, which is inferred from different types of evidence to be claimed at different levels of the design and implementation.

The third key is to note that a primary claim can be reified into *subclaims* on evidence to be provided by lower levels. Like specifications, sub-claims can themselves be recursively reified onto lower levels.

Making use of these three keys, the justification approach consists in decomposing every primary claim into its component subclaims at the first top level, and then either justify these subclaims with evidence available at this level, or to further reify the subclaim into subclaims on evidence to be provided by lower levels, and proceed in this way level by level down to the lowest.

3.2. Levels of evidence

What are these levels where evidence is to be found?

The total supporting evidence for a primary dependability claim necessarily consists of evidence of one or more of four different kinds. Without going into detail, let us say simply here that these four kinds result from the way computer systems are designed; they are also confirmed by experience.

Intuitively, it is not difficult to see that a primary dependability claim needs to be supported by one or more of the following levels of evidence:

level 1.*Plant-computer system interface*: evidence that the functional and non-functional computer system specifications are valid, that they satisfy the dependability requirements and adequately deal with the *environment/system constraints* and undesired events;

level 2.*Architecture*: evidence that the computer system architecture can support the functional and non-functional computer system specifications validated by evidence at level 1;

level 3. *Design*: evidence that the computer system is designed and implemented so as to perform according to the functional and non-functional specifications validated by evidence at level 1;

level 4. *Control of operations*: evidence that integration and operation of the computer based system *in its environment* preserves the primary claim and the environmental constraints during the whole lifetime. This may include evidence that the computer-based system can be properly maintained and cannot display behaviours unanticipated by the specifications, i.e. that behaviours outside the design basis will be detected, controlled, and their consequences mitigated.

Examples of types of evidence at each level are:

- At level 1: a regulation or a regulatory position asserting or justifying the validity of a computer system requirement specification; a safety analysis report.
- At level 2: a statement from a certification body guaranteeing certain properties of a hardware/software platform; a (set of) test result(s)s;

evidence of independency to support an architectural property such as redundancy or diversity.

- At level 3: a (set of) module(s) or integrated test result(s); a conclusion of a code static analysis or failure mode analysis; defensive programming measures.
- At level 4: maintenance procedures, periodic tests, operator control interventions or operational procedures guaranteeing a safe state or an invariant property; a (set of) pertinent operational feedback data.

3.3. Claim expansion

How should we use these levels of evidence to build a deductive argument justifying an initial dependability claim? The answer is essentially based on the fact that the four corresponding claim reification levels mentioned in section 3.1 are also levels *of causality*.

To each level correspond specific system properties, functions and also undesired events. The undesired events are those anticipated events susceptible to cause the functions at that level to fail or these properties to be invalidated.

Moreover, a function or a property usually relies on functions and/or properties implemented at lower levels. Functions and properties can thus fail also as a consequence of undesired events occurring at lower levels.

The consequence is that the justification of a claim made at a given level for a property, a correct function implementation or a proper undesired event treatment rests on evidence provided at that same level *AND on the implication (claims)* that certain properties, functions and a proper treatment of undesired event proper exist at lower levels. Making such implications is similar to claiming that lower levels are somehow correct and free of unsafe failures. Similar implications are made in hierarchical design.

These implications mean that a claim on the dependability of a function or a property at a given level usually implies claims on evidence provided by lower levels. Therefore, any initial dependability requirement needs to be expanded into primary claims at the first level. In turn every primary claim may need to be expanded into sub-claims at any of the three lower levels, and recursively, any such sub-claim may need to be further expanded into sub-claims at lower levels.

Thus, a *subclaim* is a claim made at some level i, i=1,2,3 that evidence should exist and be provided at some lower level j, j>i. When necessary, we call this a *delegation subclaim,* or in short a *delegation,* denoted α.clm[i,j], i < j. α is a name identifying the evidence claimed at level i and which must exist at level j.

Thus, as shown in Figure 3, an initial dependability requirement, a primary claim or sub-claim is the *root of a tree of implications of depth at most four*. The intermediate nodes of the tree are delegation subclaims and all end nodes (leaves) must be evidence components, otherwise the argument is not terminated. The tree of implications is the *argument* supporting the (sub)claim at the root.

As a consequence, all subclaims at a given level 2,3 or 4 must originate directly or indirectly from a primary claim at level 1.

3.4. Layered Arguments

Concretely, an argument must in general consist of the following types of implications.

Level 1: Plant – Computer System Interface:

Primary Claims:

At this level, an initial dependability requirement needs to be expanded into a necessary and sufficient set of functional and non-functional primary claims.

Evidence:

The functional primary claims claim properties of the computer system functional specifications such as their validity, completeness, and compliance with the initial dependability requirement. They may also have to claim that the undesired events that can occur in the environment and the corresponding safe states are properly specified. Functional diversity may also have to be claimed at that level. All these functional primary claims must be exclusively supported by evidence of the same level 1.

Delegation:

The non-functional primary claims address properties of the implementation such as reliability, availability, correct, fault tolerant and fail-safe implementation of the specifications. These non-functional primary claims must be exclusively supported by evidence at the lower levels of architecture, design and control. At level 1 these non-functional primary claims actually infer properties of these lower levels; these properties are assumed satisfied at level 1 and their justification is delegated by subclaims to lower levels.

Level 2: Architecture

Each delegation sub-claim made at level 1 on level 2 assumes properties of the architecture and must be supported by level 2 evidence and/or expanded into delegation sub-claims to lower levels.

Evidence

Evidence to support delegation subclaims on sensor and actuation devices, fault tolerance, redundancy, hardware diversity, physical separation, communication links and protocols, response times, mapping of software on hardware architecture…

Delegation

Subclaims on protocol and system platforms, on software correctness, fault tolerance, fail safeness, time performances, self-diagnostics, on COTS behaviour…

Level 3: Design

Each delegation sub-claim made at level 1 and 2 on level 3 assumes properties of the design and must be supported by level 3 evidence and/or expanded into delegation sub-claims to level 4.

Evidence

Evidence on application, communication and system software correctness, testability, performances, fail safeness…

Delegation

Subclaims on periodic tests, operator alarm and controls, in-service and maintenance procedures…

Level 4: Operation and Control

Each delegation sub-claim made at level 1, 2 and 3 on level 4 assumes properties of the control and must be supported by level 4 evidence.

Evidence

Evidence on operator, in-service and maintenance procedures, periodic tests…; from operational feedback.

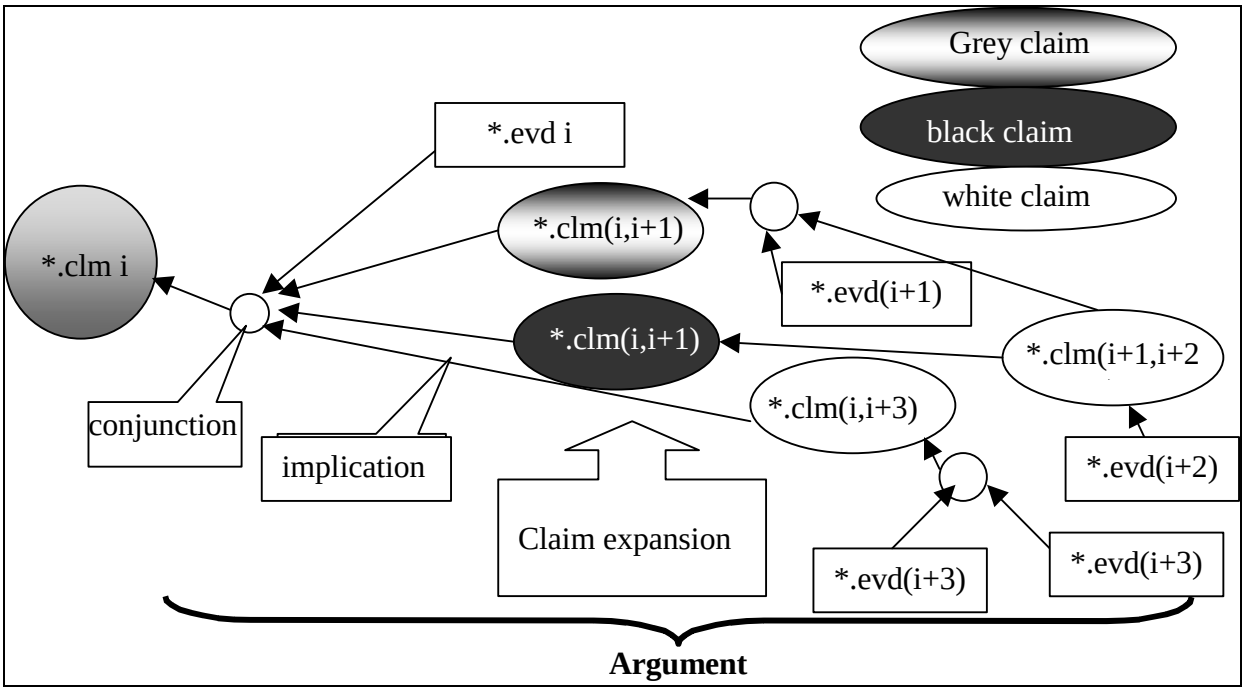

Figure 3

As an example, Figure 4 shows the argument supporting the initial dependability requirement "no rotation" which was introduced in section 2; this requirement is denoted "norot.clm0" at the top of this figure. The expansion illustrates the complexity of the justification of an apparently simple claim.

4. Argument Properties

A sub-claim can therefore be of one of three different types. Using colours (see Figure 3) to refer to these types more easily:

- *White*: the subclaim is exclusively supported by evidence at the same level.
- *Black*: the subclaim is exclusively inferred from lower level by delegation sub-claims,
- *Grey*: the subclaim is supported by evidence at the same level *and* also by delegation sub-claims.

A black or a grey subclaim is a "*logical link* " across two levels of evidence: the level at which it is formulated and assumed true, and the level at which it is proven true. Or, in other words, a black or grey subclaim is a logical construct to relate the level at

which evidence is found to be needed, and the level at which this evidence is actually provided.

An initial dependability requirement can be considered as a black claim.

At level 1, there is no grey claim. A primary claim is either white (i.e. functional) or black (i.e. non-functional).

At intermediate levels 2 and 3 expansions are arbitrary and claims can take any colour.

At the lowest level 4, subclaims must all be white.

4.1 Safety Justification is a Reverse Inductive Process

Safety justification appears to be a *backward or reverse inductive* process, rather than a direct inductive or deductive process[3]. This is another difficult aspect of the demonstration of safety. Criminal and police officers, as inspectors Maigret and Poirot, make direct inferences. They start by collecting evidence, if possible in absence of any presumption, until enough is accumulated to argue that someone is guilty. In safety justification, we have to start by stating what are the safety requirements and properties to justify, and only then construct arguments and identify what evidence is necessary. It would be unacceptable and dangerous to work the other way round, as it would somehow mean adapting the safety claims to the available evidence.

To paraphrase rules for specifying software requirements [9]: state the "safety issues" before answering them. If this is not done, the available evidence may prejudice the safety claims so that the easily justified ones only are claimed.

4.2 Single argument and conjunctive property

The previous sections show that at any level i of an argument, a (sub)claim α is implied from a *conjunction* of evidence components $\upsilon,\omega,...$ from the corresponding level *i* and/or from delegation subclaims $\beta,\gamma,\ldots$ on lower levels:

$$\alpha.\text{clmi} \Leftarrow \{\upsilon.\text{evd[i]} \wedge \ldots \omega.\text{evd[i]}\} \wedge \{\beta.\text{clm.[i,j]} \wedge \gamma.\text{clm[i,k]} \wedge \ldots\}$$

where levels j, k,…> i.

(eq 1)

It is important to understand why the right hand side of this implication is always a *conjunction* (i.e. an AND) of (sub)claims and of evidence components. A disjunction (OR) would mean that there are alternative arguments to support a subclaim and this possibility should be discarded for the sake of completeness. One must indeed distinguish two cases. Either one of the alternatives would be sufficient to support the claim. In this case the safety justification would be simpler by just retaining that argument. Or none of the alternative arguments is sufficient by itself (as for instance in a three leg argument). Then, all these alternative arguments are necessary. In this case we must have a conjunction of antecedents in a single argument.

[3] It is for the sake of simplicity that the title of the paper refers to a "deductive" approach.

As a concrete example, and to grasp the full significance of this conjunctive property, consider some deterministic[4] claim at architecture level 2 that would assert that a processing unit is fail-safe in its output behaviour.

This claim might be inferred from either one of two subclaims at design level 3:

(i) A subclaim that a failure mode and consequence analysis of the software of this processing unit has identified potential errors and has rightly concluded that the software failures are safe,

(ii) A subclaim that a hardware maximum cycle timer and the self-tests of the processing unit trap potential errors caused by this unit and leave the system in a safe state.

Then, three possibilities arise:

1. Each of these two subclaims is shown with supporting plausible evidence to cover the whole set of potential errors. In this case (in practice unlikely), only one claim (the most plausible one) is necessary.
2. Neither the software FMCA analysis nor the hardware timer can be claimed to cover alone the whole set of potential errors, but the two subclaims justifiably (i.e. with supporting plausible evidence) complement each other by addressing complementary sets of potential errors. This is clearly a conjunction of the two subclaims together with a sub claim and evidence on their complementarities.
3. Each subclaim intends to cover the complete set of potential software errors, but with a certain degree of uncertainty (probability). In this case, the two subclaims together are intended to re-enforce statistical confidence in the completeness of the coverage of potential errors, on the basis, for instance, that they use independent and different means of detection and protection. In this case, the claim on the failsafeness of the unit must be stated as a probabilistic one, and must be clearly inferred from a conjunction of the two sub-claims, also expressed as probabilistic claims, together with a sub claim and evidence that the assumption of independence and complementarity is correct.

Thus, when two or more subclaims – together – or two or more pieces of plausible evidence are intended to mutually re-enforce confidence, on the assumption that they are independent and of different nature, then claims and evidence are required to explicitly assert and justify this mutual re-enforcement, their independence and their differences.

More generally, a primary claim or a subclaim is always supported by one argument only. In particular, a sub-claim that appears in the right-hand side of more than one expansion must be supported with the same argument. This univocal property is a necessary condition for the consistency of the safety justification, and also for allowing sub-claims to be reused with their arguments.

[4] A similar example could be conceived with a probabilistic claim.

5 Models and Documentation

It is wrong to think that Physics is about what Nature is.
Physics is about what we have to say about Nature.

Niels Bohr

In physics and in engineering, models are simplified representations of reality that highlight some aspects and ignore others. They are however indispensable as they enable us to analyse, to reason, to evaluate and to communicate about real world systems. They can be source of misunderstandings and misinterpretations if they remain implicit or not properly defined.

Models are essential in system and software development. They are equally important for the demonstration of system safety; in at least four different practical ways:

1. Safety, reliability, availability and other attributes of a system can only be apprehended by means of observations and by models, the latter being necessary to give meaning and purpose to the former,

2. Models are necessary to formulate and define the semantics of claims, evidence, and arguments,

3. Models are necessary to establish properties essential to safety such as completeness (cfr. Section 2.4),

4. Models of the system and of the safety justification provide a necessary basis for defining and supporting the documentation required for the safety case.

Surprisingly, models have been rather neglected by engineers and researchers working on computer system safety cases. Exceptions perhaps are the following recommendations of the IAEA safety guide NSG 1.1 [1]:

> 3.17. The requirements for and design of the software for systems important to safety should explicitly define all relations between input and output for each of the operating modes. The software design should be simple enough to permit consideration of all input combinations that represent all operating modes.
>
> 5.14. There should be a precise definition of the system boundaries, i.e. of the interface between the computer based system and the plant. In particular, the interfaces of the system with the sensors and actuators, the operator, the maintainer and any other external system should be specified.
>
> 7.14. Software requirements may be based on a model of the system to be implemented (Section 5, and Section 5 of Ref. [4]). In this case the model and its application should be well defined and documented with specification of the requirements. For example, control software is sometimes described using a finite state machine model. The use of the finite state machine model should be described so that the requirements for state transitions and functions specific to particular states can be properly understood.

5.1 The Four Layered Models

The justification framework discussed in the previous sections gives indications on the models that would be needed and on the safety justification documentation that should be made available.

As the four levels of reification at which claims and evidence must be formulated address quite different aspects of the system, a different model is needed for every of the four levels:

a model of the interface between the system and its environment
a model of the system architecture
a model of the hardware and software design and implementation;
a model of the system modes of use, operator and maintenance controls.

At each level, the model must be a precise description of:

- the *assumptions* and the *constraints* imposed on the system entities at the corresponding level; that is assumptions and constraints imposed by the plant environment (level 1), by the computer and other existing equipment architecture (level 2), by the hardware and software design and technology (in particular the COTS and the software platforms used) (level 3), and by operation controls and procedures in place (alarms, operator interactions, maintenance, periodic tests…) (level 4).

- the *functionality* of the system, i.e. of the interactions required and expected from the system with its environment at that level,

- the *undesired events* that may occur at each level; not only the normal behaviour of the environment, but also the hazards, accidents and failures that may affect the system environment, its hardware and software architecture, its implementation, and the operation control (e.g. the human operators) must be part of the design basis of a system important to safety. These "*undesired events* must be anticipated and must be part of the model at the corresponding level.

Functional relations between environment and system state variables provide a single and convenient modelling tool to represent these *assumptions, constraints, system functions, and undesired event consequences*.

Four level models that use this approach are proposed and worked out in [6]. This modelling work cannot be described in detail here. It is based on the seminal models developed by D.L. Parnas (see for instance [10],[9]). Relational models, already used in Darlington [10], work well for control systems. It is shown in [6] how it can be applied to safety justification. Figure 5 gives a succinct view of the functional relations developed in [6] for each level, and a short description of the functional relations of these models is given in the appendix.

Such models can constitute a solid basis for structuring the argumentation and documentation of safety cases.

5.2 Model Properties and Inter- Relations

Thus, a safety justification turns out to be a logical analysis based on at least four types of models, which are quite different from each other. This is another great difficult aspect of the safety demonstration.

To carry out the arguments from the top level down to the lowest, these models cannot be independent from each other. To understand how they should be interrelated, the key point is to remember (cfr. Section 3.3) that a (sub)claim is part of an implication at two distinct levels. A sub-claim is a right antecedent of an implication such as (eq 1) at some level and the left consequent of such an implication at some other level below. In other words, a subclaim is made at some level and proved by evidence at some other level below.

Therefore models at all levels must be *consistent* so as to provide the ability of formulating consistent claims and evidence at different levels.

The models briefly sketched in the appendix illustrate the relations that must exist between them to carry over the arguments from the level 1 at which the primary claims are expressed, through the implementation levels down to the operator and maintenance control level.
In terms of model theory concepts (see e.g. [4]), each model is an *extension* of an upper level model *substructure* and a *substructure* of the next lower level model *extension*. Such a hierarchy has in particular the following model properties [4] between two adjacent levels:

(i) The domain of an *extension* contains the domain of its upper level *substructure*.

(ii) The functional relations of a *substructure* are the *restrictions* to its domain of the relations of its lower level *extension*; that is the relations of a *substructure* must also be relations in the lower level *extension*.

(iii) All free variables in a model *extension* are assigned values from the domain of the upper level *substructure* from which it is extended.

6. Concluding Remarks

The purpose of this stratified justification approach is to start from the initial safety requirements of an application and to demonstrate, in as simple a form as possible, but not simpler, their correct implementation at the different levels of a complex design. We do not claim that safety justifications are simple. Proofs do not permit to ignore or even to leave implicit the complexity of the arguments. Our objective is to try to document and control this complexity by means of structures and tools.

The main technical features of the approach are:

- The use of a four-layered structure for organising evidence, claims and arguments, models and safety case documentation.

- Concepts and mechanisms for the construction of arguments: inductive expansion of claims, conjunctive implications, and delegation of evidence onto lower layers.

Expected benefits are:

- The advantages of a goal/claim based approach; in particular to limit the required evidence to what is *arguably* necessary and sufficient;

- Better structuring of arguments for restricting and backing up subjective expert judgement;

- Easier independent verifications of arguments;

- More natural and easy transformations of non-functional claims into functional ones;

- Possibilities of assessing the weight of a particular component of evidence in the whole justification,

- Modular re-use of arguments, of subclaims and sub-safety cases for integration of platforms, sub-systems and pre-existing software.

The latter two advantages, not discussed in this paper, are investigated in [5].

This safety justification approach attempts to give precedence to - and to focus on - the safety properties of the system behaviour, and by the same token to offer the possibility of being cost effective in terms of efforts and resources spent on the justification: two objectives which meet priorities of both regulators and licensees. Meeting their priorities should hopefully also help to make their negotiations more supple and efficient.

7. References

1. IAEA Safety Guide N° NS-G-1.1 *Software for Computer Based Systems Important to Safety*. INTERNATIONAL ATOMIC ENERGY AGENCY VIENNA, September 2000.
2. IEC 643. Application of Digital Computers to Nuclear Reactor Instrumentation and Control. 1st edition, 1979.
3. Courtois P.-J., Parnas D.L. Documentation for Safety Critical Software. IAEA Specialists' Meeting on *Software Engineering in Nuclear Power Plants: Experience, Issues and Directions.* AECL Research, Chalk River Laboratories. Chalk River, Ontario, Canada. September 1992. Reproduced in IEEE Proceedings of 15th International Conference on Software Engineering, Baltimore, May 1993.
4. Courtois P.-J. Semantic Structures and Logic Properties of Computer-Based System Dependability Cases. Nuclear Engineering and Design 203 (2001) 87-106.
5. Courtois P.-J., *Hard Guidelines Made for Computer Software.* Nuclear Engineering International, January 2002, Vol 47, N° 570, pp.37-40.
6. Courtois P.-J. A Framework for the Safety Justification of the Dependability of Computer-Based Systems, October 2004; available from http://www.avnuclear.be/avn/dependability_framework.pdf
7. Feynman R.P. "Cargo Cult Science: Some remarks on science, Pseudoscience, and Learning How to Not Fool Yourself". The 1974 Caltech Commencement Address. Reprinted in "*The pleasure of finding Things Out*", J. Robbins ed., Penguin Books, 1999.
8. Govaerts, P. Possible Use of Reliability and Safety from other technologies. Proceedings ESA Symp. "*Ground data Systems for Spacecraft Control"*. Darmstadt, FRG, June 1990.
9. Heninger, K.L., Specifying Software Requirements for Complex Systems: New Techniques and their Application, IEEE Transactions Software Engineering, Vol.

SE-6, No. 1, January 1980, pp. 2-13. Reproduced in *Software Fundamentals-Collected Papers by D.L. Parnas*, Edited by D.M. Hoffman and D.M. Weiss, Addison –Wesley, 2001, pp. 387-392.
10. Parnas D.L., Asmis G.J.K., Madey J., 1991, Assessment of Safety Critical Software in Nuclear Power Plants. *Nuclear Safety*, 32, 2.
11. D. Pavey, R. Bloomfield, P.-J. Courtois et al. Cost Effective Modernization of Systems Important to Safety (CEMSIS). Proceedings of FISA-2001. EU Research in Reactor Safety. Luxemburg, 12-15 November 2001. EUR 20281. pp.213-225. ISBN 92-894-3455-4.
12. Saglietti F. Licensing Reliable Embedded Software for Safety-Critical Applications. Real-Time Systems, 28, 217-236, 2004.

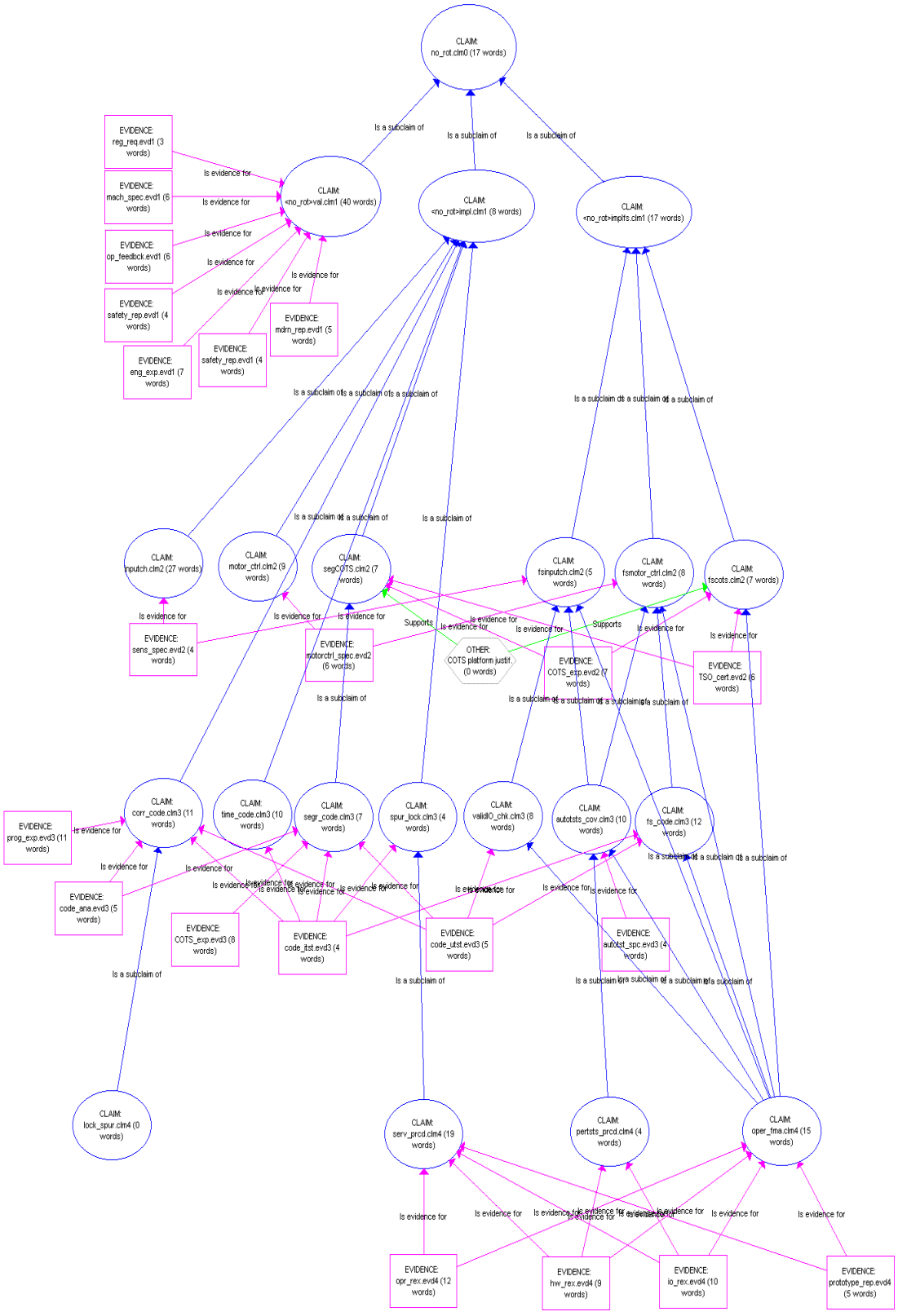
CLAIM: no_rot.clm0 (17 words)
CLAIM: <no_rot>val.clm1 (40 words)
CLAIM: <no_rot>impl.clm1 (8 words)
CLAIM: <no_rot>impls.clm1 (17 words)
EVIDENCE: reg_req.evd1 (3 words)
EVIDENCE: mach_spec.evd1 (6 words)
EVIDENCE: op_feedbck.evd1 (6 words)
EVIDENCE: safety_rep.evd1 (4 words)
EVIDENCE: eng_exp.evd1 (7 words)
EVIDENCE: safety_rep.evd1 (4 words)
EVIDENCE: mdrn_rep.evd1 (5 words)
Is a subclaim of
Is evidence for
CLAIM: inputch.clm2 (27 words)
CLAIM: motor_ctrl.clm2 (9 words)
CLAIM: segCOTS.clm2 (7 words)
CLAIM: fsinputch.clm2 (5 words)
CLAIM: fsmotor_ctrl.clm2 (8 words)
CLAIM: fscots.clm2 (7 words)
EVIDENCE: sens_spec.evd2 (4 words)
EVIDENCE: motorctrl_spec.evd2 (6 words)
OTHER: COTS platform justif. (0 words)
EVIDENCE: COTS_exp.evd2 (7 words)
EVIDENCE: TSO_cert.evd2 (6 words)
Supports
CLAIM: corr_code.clm3 (11 words)
CLAIM: time_code.clm3 (10 words)
CLAIM: segr_code.clm3 (7 words)
CLAIM: spur_lock.clm3 (4 words)
CLAIM: validIO_chk.clm3 (8 words)
CLAIM: autotsts_cov.clm3 (10 words)
CLAIM: fs_code.clm3 (12 words)
EVIDENCE: prog_exp.evd3 (11 words)
EVIDENCE: code_ana.evd3 (5 words)
EVIDENCE: COTS_exp.evd3 (8 words)
EVIDENCE: code_itst.evd3 (4 words)
EVIDENCE: code_utst.evd3 (5 words)
EVIDENCE: autotst_spc.evd3 (4 words)
CLAIM: lock_spur.clm4 (0 words)
CLAIM: serv_prcd.clm4 (19 words)
CLAIM: pertsts_prcd.clm4 (4 words)
CLAIM: oper_fma.clm4 (15 words)
EVIDENCE: opr_rex.evd4 (12 words)
EVIDENCE: hw_rex.evd4 (9 words)
EVIDENCE: io_rex.evd4 (10 words)
EVIDENCE: prototype_rep.evd4 (5 words)

Figure 4

8. APPENDIX

Model Functional Relations of Figure 5

At level 1, environmental variables are either *monitored* or *controlled; monitored* variables **m(t)** are those that need to be measured by the system; *controlled* variables **c(t)** are those that the system is intended to control. These variables have values that can be a function of time. **FS(t)** is the the sub-set of **c(t)** values that correspond to fail-safe states of the environment.

NAT is the set of relations dictated by the constraints of the environment between the values of the **m(t)** and **c(t)** variables. ***REQ*** is the set of functional relations between the values of **m(t)** (***REQ*** domain) and the values of **c(t)** (***REQ*** range) that specify the required system behaviour; ***REQ*** is thus the specification of the system requirements.

$\mathbf{h^1(t)}$ is the vector of entities in the environment structure that are potential sources of undesired events that are postulated, e.g. a failed motor, a pipe break, or a malfunctioning valve. An element of $\mathbf{h^1(t)}$ indicates whether an undesired event has occurred and which type. The occurrence of an undesired event at time t is described by the pair $(h^1(t),t)$. There must be a set of relations that descrie how the variables monitored by the system are affected by these postulated undesired events. $\boldsymbol{HAZ^1}$ describes this set of relations.

At level 2, for each channel k, two sets of variables are identified: a set of inputs, variables $\mathbf{i_k(t)}$ that can be read by the channel, and a set of output variables $\mathbf{o_k(t)}$ whose values are determined by the computers of the channel. These variables are associated with input registers (data acquisition) and output registers (actuators) of the channel; their values are also described by time-functions.

Undesired event that may occur at architecture level, and are part of the design basis must be observable by the architecture. $\mathbf{h^2(t)}$ is the vector of architecture entities that are potential sources of undesired events to be considered and postulated, e.g. a failing sensor, a failing input or output register, a communication line; $\boldsymbol{HAZ^2}$ is the set of relations that describe how undesired events $\mathbf{h^2(t)}$ may affect the input and output channels.

At level 3, the software should provide a channel k with input-output behaviour that can be described by a relation $\boldsymbol{SOF_k}$, the domain of which is a set of possible values of $\mathbf{i^k(t)}$, and range is the set of the possible values of $\mathbf{o^k(t)}$.

Undesired events may affect the *processing hardware or the software*. $\mathbf{h^3(t)}$ is the vector of the potential causes of postulated *hardware* undesired events, e.g. a processor failure, a failing register, a failing memory location, a numerical or memory under/overflow, a buffer or stack over/under flow, an interrupt caused by a software check or an invariant violation, a watchdog run out time,… There is a set of relations between the values taken by the channel state and these undesired events. $\mathbf{s_k(t)}$ is the vector of all states that the channel k processing unit (processor and memory) can be in and $\mathbf{HAZ^3_k}$ the set of relations that define how undesired events affect the channel state.

Software failures are undesired run-time events of different type. They are the consequences of errors caused by anomalies and defects of the software that escaped the verifications. They correspond to relations $\mathbf{SOF_k}$ that would not bet acceptable, i.e. that do not to comply with the relations **REQ**. Evidence on the acceptability of the relations $\mathbf{SOF_k}$ must be provided. More precisely, for each channel k, one must have:

$$\mathbf{REQ_k \Leftarrow IN_k \wedge SOF_k \wedge OUT_k \wedge NAT} \qquad \text{(eq 2)}$$

At level 4, control of operation (human operators, procedures, maintenance, periodic tests…) can be viewed as a channel, which is external to the system and is part of its environment. This channel receives information from the environment and from various parts of the system. It can a send information to and re-act upon different parts of the environment and the system. The operation control channel is supposed to have direct reading and writing access to input registers $\mathbf{i^k(t)}$ of the channels, reading access to output registers $\mathbf{o^k(t)}$, but no access to the internal state $\mathbf{s^k(t)}$ of each channel. This control channel is at any time in a distinct state defined by the value of a vector $\boldsymbol{ctr(t)}$.

Relations $\boldsymbol{READ_k}$ and $\boldsymbol{REACT_k}$ specify the behaviour of the operators, users and maintenance team when they follow the operational procedures, the maintenance, calibration and manual control procedures to capture and intervene on the state of channel k. Undesired events $\mathbf{h^4(t)}$ that may occur at this level are human failures, external postulated initiating events, potential defects in control and maintenance procedures, etc….

For a detailed analysis of these models, see [6].

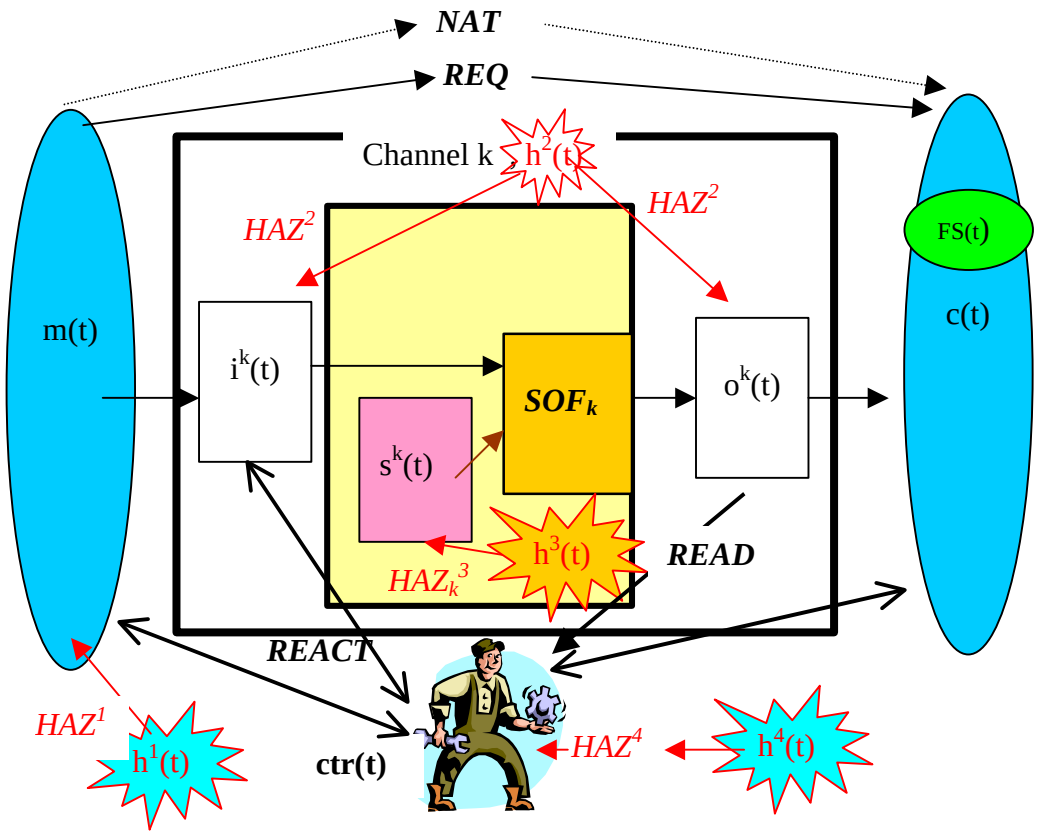
NAT
REQ
Channel k
h2(t)
HAZ2
HAZ2
FS(t)
m(t)
c(t)
ik(t)
SOFk
ok(t)
sk(t)
h3(t)
READ
HAZk3
REACT
HAZ1
h1(t)
ctr(t)
HAZ4
h4(t)

Figure 5