ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects

O. Akerlund[1], P. Bieber[2], E. Boede[3], M. Bozzano[4], M. Bretschneider[5], C. Castel[2], A. Cavallo[6], M. Cifaldi[7], J. Gauthier[8], A. Griffault[9], O. Lisagor[10], A. Lüdtke[3], S. Metge[5], C. Papadopoulos[5], T. Peikenkamp[3], L. Sagaspe[2], C. Seguin[2], H. Trivedi[5], L. Valacca[7]

1 Prover Technology AB, Rosenlundsgatan 54, 118 63 Stockholm, Sweden
2 ONERA, 2 avenue E. Belin, 31055 Toulouse, France
3 OFFIS, Escherweg 2 - 26121 Oldenburg, Germany
4 Istituto Trentino di Cultura, Via Sommarive 18, Povo, 38050 Trento, Italy
5 Airbus France, Germany, UK
6 Alenia Aeronautica S.p.A., Strada Malanghero 17, IT-10072 Caselle, Turin, Italy
7 Societa' Italiana Avionica S.p.A., Strada Antica di Collegno, 253 - 10146 Turin, Italy
8 Dassault Aviation, Saint-Cloud, France
9 LaBri, Université de Bordeaux, France
10 Department of Computer Science, The University of York, YO10 5DD, UK

Abstract: This paper aims at presenting methods and tools that are developed in the ISAAC project (Improvement of Safety Activities on Aeronautical Complex Systems, **www.isaac-fp6.org**), a European Community funded project, to support the safety assessment of complex embedded systems. The ISAAC methodology proposes to base as much of the safety analyses as is feasibly possible on simulable and formally verifiable system models that include fault models and can be shared both by safety and design engineers. On one hand, tools were developed to support safety assessment of Simulink, SCADE, Statemate, NuSMV and AltaRica models. On the other hand, formal models are coupled with additional models to address the problems of common cause analysis and human error analysis.

Keywords: system safety assessment, certification, formal methods

1. Introduction

This paper aims at presenting methods and tools that are developed in the ISAAC project (Improvement of Safety Activities on Aeronautical Complex Systems), a European Community funded project, to support the safety assessment of complex embedded systems.

Typical avionic systems studied in ISAAC are heterogeneous systems made up of computer-based controllers (e.g., real-time controllers) and mechanical components. Moreover, for safety and operational reasons, these systems are often reconfigurable, thus making it difficult to master the numerous combinations of cases and the system dynamics during the safety analysis activity. ISAAC aims at improving the classical safety analysis means, such as fault tree analysis, which are devised to cope with the system structure without taking into account the system dynamics. Furthermore, ISAAC wants to improve the sharing of information, which is specific to safety engineers, with system designers.

These objectives have been partly tackled by a previous European project called ESACS (Enhanced Safety Assessment for Complex Systems, 2001-2003). The ESACS project proposed to base the safety analysis on dynamic formal system models that include fault models and the ESACS teams used more specifically the following formal languages for modelling reactive systems: Simulink, SCADE, Statemate, NuSMV and AltaRica [1]. Safety assessment tools were developed based on existing proof engines. On the one hand, these tools enable to search for sequences of events that lead to failure conditions of interest. On the other hand, they enabled the assessment of qualitative safety requirements such as no single failure shall lead to a critical failure condition. The ESACS methodology and tools were applied to several case studies provided by the aircraft manufacturers.

On the basis of these promising first results, the ISAAC project aims at generalizing the use of numerical simulable models to support a more comprehensive set of safety analyses. On the one hand, new tools are being developed to extend the use of formal models on their own. In particular, they support safety requirement allocation, failure mode and effect analysis, testability analysis and mission reliability analysis. On the other hand, formal models are coupled with additional models to address the problems of common cause analysis and human error analysis. In the case of common cause analysis, common cause failures, due to the

geometrical layout, are found through the use o f 3D models (e.g., CATIA models) and are injected into the formal model that specifies the functional behaviour of the system. Similarly, in the case of human error analysis, plausible pilot errors are investigated by coupling flight simulators and models of pilot mental states and operation procedures. Then, the emerging pilot errors are injected in the formal model to take into account not only faults due to the physical components, but also the human factor.

In this paper, we give an overview of the comprehensive set of methods and tools developed so far and we sum up the lessons learnt after the application of our approach to several significant case studies.

2. Building simulable formal models of complex system and their failures

The first step of the ISAAC methodology is to build a formal system model that includes fault models with one of the following notations: Simulink, SCADE, Statemate, NuSMV and AltaRica. Two ways of building such formal failure propagation models were envisaged. In the first scenario, models are specially built for safety analysis purpose. So details regarding the normal behaviour are not modelled but rather a nominal abstract behaviour is modelled allowing for the focus to be placed on failures and their propagation. In the second scenario, formal design models exist because they were provided to support system functional analysis. They depict the expected system behaviour. These models are extended to deal with the failure propagation.

The following sections show how the ISAAC methodology and tools support both scenarios.

2.1 Failure propagation models based on predefined components

In the first scenario, we propose to assist the construction of a failure propagation model with the use of predefined libraries of formal components. The library of components is a collection of formal models of basic system components that are immediately suitable for safety analysis. Thus, each formal model describes both nominal and faulty behaviour of a specific system component and the model granularity is defined according to the safety analysis purpose.

In the earliest phases of the system design, details of physical components are not fixed and safety engineers work with simple functional blocks and functional failure modes, (eg. function permanent loss). Usually, a block offers a service (outputs) provided that some inputs and resources are available in the nominal case (see figure in section 4.1). Only permanent failures are considered and, after a failure, the block no longer provides an output. Some other blocks do not require inputs or resource to play their role (e.g., source of energy), or they do not provide any output (receptor). The library contains a first family of such generic blocks.

During the Preliminary System Safety Assessment (PSSA, [2], [3]), the system architecture is developed, there is a greater understanding with regards to the behaviour of components, hence safety engineers take into account more specific failure modes. At this stage, the library currently contains two other families of components. One is dedicated to components of hydraulic systems (reservoir, pump, valve, pipe, …) and deals with failure modes such as total loss, leak in a component, leak propagation, … Second is dedicated to components of electrical systems (generator, bus, switch, receptor, …) and handle failures such as total, short-circuit, … A third family of components is under development to deal with a flight control system.

Whatever are the development phases, all these failure propagation models are qualitative ones. For instance, inside the hydraulic library, three levels of fluid (empty, low, normal) are considered instead of a real value, that measures the fluid pressure. The library is currently written in the AltaRica language [4]. Basically, each AltaRica model consists of two parts. An automaton describes which failure or nominal mode may be activated when a failure or a normal event occurs. Then, a set of logical assertions describes the relationship between the input/output of the components according to the current modes (see for instance [5]). The library is implemented within the Cecilia-OCAS graphical toolkit. A translator from AltaRica to Lustre language has been developed[1] and will allow the ability to have similar libraries written in the Lustre language, usable in the SCADE environment

2.2 Failure mode extension

In the second scenario, the process is initiated by the design engineer, who provides a formal model of the design, the System Model (SM for short), at a given level of granularity. Initially the model includes only the nominal behaviour of the system, that is, all the components of the system are assumed to behave as expected. This model may be used by the design engineer to verify the functional requirements of the system, and it is then passed to the safety engineer for safety assessment.

The validation of the SM with respect to the safety requirements is performed by assessing the system behaviour in degraded conditions. To this aim, the

[1] See http://altarica.labri.fr/Tools/AltaLustre/

safety engineer enriches the SM with the definition of the failure modes, that is, a specification of how the various components of the system can fail. This step yields a model, that we call Extended System Model (ESM for short). The ESM is an executable specification of the design model in which all the components of the SM can fail according to the specified failure modes. This step of the process is called Failure Mode Injection, and it is performed *automatically* through an extension facility.

The failure mode types to be injected into the SM can be stored and retrieved from a library of generic failure modes, the so-called Generic Failure Modes Library (GFML for short). This library contains the definition of the failures that can be attached to system model variables. Examples of typical failure modes are, e.g., *stuck at* (output stuck at a particular value), *inverted* (corruption of a Boolean output), *ramp-down* (decrease of a given amount at each execution step, down to a given value), *random* (non deterministic output), *noise* (a random offset in a given range is added to the nominal output). Attaching a failure mode to a system variable may require specifying additional parameters that affect the behaviour induced by the failure (e.g., the value which the variable is stuck at). The GFML may be extended to include user-defined failure modes.

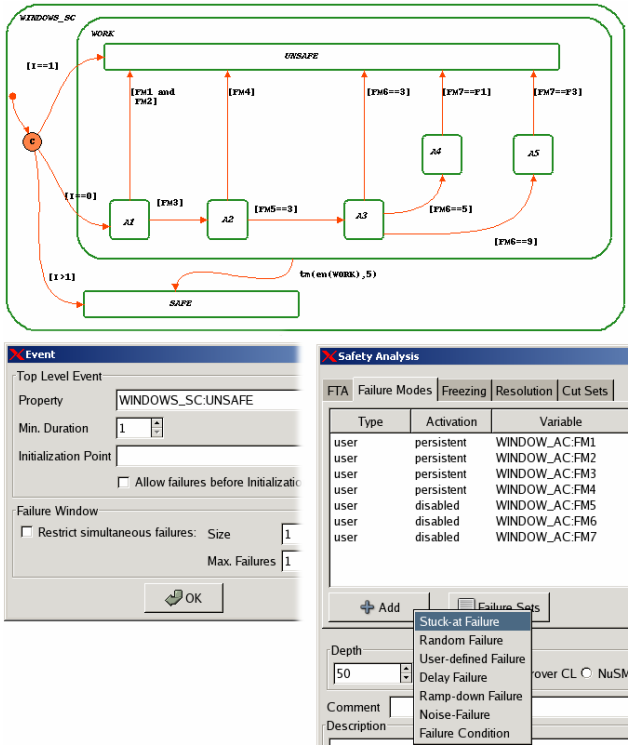

Figure 1: Safety task consisting of system model, safety requirement and injected failures.

The scenario based on failure mode extension is supported by the SCADE-based, Statemate-based and NuSMV-based platforms. As an example figure 1 shows a (part of a) Statemate design together with a safety requirement definition (on the left) and the injected failures (on the right). The SM together with the safety requirements and the injected failures constitute a *safety task*. Grouping these three types of information together allows for easy re-evaluation of a safety assessment in case of design changes. An application of the method to a Flap Control System is given in [6].

3. Generic safety assessment activities revisited thanks to formal behavioural models and associated tools

We present now how to formally check whether a failure propagation model meets a safety requirement. Typical safety requirements are qualitative ones such as “no single failure shall lead to a failure condition” or quantitative ones such as the “the probability rate of the failure condition shall be less than 10-9 per flight hour”.

Traditionally, safety analysts use a deductive approach to assess such requirements: starting from a failure condition (e.g. “function loss”) as top level event, they progressively built a Boolean tree or a diagram that catches the combination of elementary events that caused the top level event. Then fault tree tools can compute the probability rate of the top level event to assess the quantitative requirements. They can also compute the set of minimal combinations of elementary events that lead to the top level to assess qualitative requirements.

With the ISAAC approach, the basic event dependencies are stored in the formal failure propagation model independently from any top level event. We use two basic techniques to exploit the formal models: model checking and generation of fault trees or sequences of events. Model checking enables to proof qualitative requirements whereas the generation tools extract from the model the set of causes of a failure condition. Let us give more details about these two techniques.

3.1 Model checking

Model checking is one method for the exhaustive verification of embedded reactive systems [7]. It helps to find bugs that are difficult to find by testing, since they tend to be non-reproducible. This verification approach requires having at disposal three elements:

1. A system model SM. It is described within a framework for modelling the reactive system, e.g. description languages such as Statemate[2], SCADE[3].

[2] see: http://www.ilogix.com/
[3] see: http://www.esterel-technologies.com/

2. A safety requirement Req. It is formally written in a specification language for describing the properties to be verified.
3. A verification engine to establish whether the description of the system satisfies the specification.

In the ISAAC context, the system model SM (1) is built as described in section 2. Whatever is the modelling framework, SM can be considered as a transition system that depicts how the system evolves from state to state. SM can also be viewed as streams of values for a period of time steps.

The specification Req (2) is usually given in some logical formalism. It is common to use temporal logic. The idea of temporal logic is that a logic formula is not statically true or false in a model SM. Instead the models contain several states and a formula can be true in some states and false in others. Thus, the static notion of truth is replaced by a dynamic one, in which the formulas may change their truth values as the system evolves from state to state.

The model checker (a programme) (3) is applied to the system *SM* and the property *Req* to be verified. It outputs the answer ”yes” if *SM satisfies Req* (the definition of “satisfies” depends on the semantics of the temporal logic [7]) and “no” otherwise; in the latter case the model checker produces a trace of the system inputs which causes the violation of Req. This automatic generation of such “counter-examples” (CEX) is an important tool and will be applied in the following chapters.

3.2 Fault tree and sequence

Let ESM be a formal failure propagation model that depicts how system variables X1, X2, … evolve in the course of time in nominal mode or when a failure mode FM1, FM2, … is active. Let SR be a safety requirement expressed by some temporal formula. Let TLE be the top level event corresponding to the violation of SR. Then, the cause (fault-tree or sequence) generation analysis aims at finding out which combinations of FM can cause the violation of SR. The computation is based on the counter-example (CEX) generation of the proof engine previously described. One tries to prove that ESM fulfils SR. If ESM violates SR, then each CEX is a possible cause.

Thus, the analysis is done in the same fashion as traditional fault tree analysis (FTA), - i.e. one identifies minimal combinations of failures (Minimal Cut Sets) violating SR - but with the difference that traditional FTA is static whereas the ISAAC approach can also include temporal aspects. The figure 2 illustrates this difference.

Notice that input and output variables to ESM and SR represent streams of values for a period of time steps. This makes it possible to represent analysis results, see Figure 2, showing a sequence of values for each variable. When SR is violated it is of special interest to notice the FM’s, which are included in the CEX.

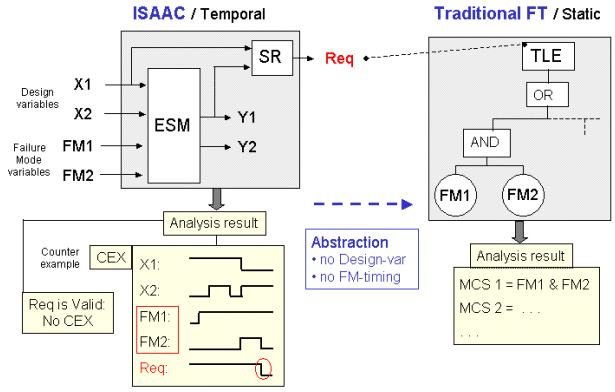

Figure 2 Temporal vs. static methodology

If there is no further interest in the CEX than observing the FM’s – i.e. no attention is paid to which design variables (X) are included or to any sequences – the analysis result is interpreted in a traditional “static” way. This interpretation can be seen as an abstraction of the temporal approach and is illustrated in the right part of Figure 2.

Taking also the temporal information into account it is possible to perform a number of analyses, e.g. importance of ordering among FM’s, duration of FM, influence of system state, etc. The CEX shown after an execution represent *one* possibility to violate SR and therefore it is of interest to concentrate and elaborate on the cut set found.

Looking at Figure 2 we see e.g. that FM1 occur before FM2 and we can be interested to know if their ordering is of importance. This type of analysis is done automatically by the ISAAC platforms.

In the CEX, in Figure 2, FM1 occurs (becomes true) at an early time step and remains true permanently. ISAAC platform makes it possible to induce FM1 to become transient, i.e. once it becomes true it is only true for a few time steps, which is decided by the user. If analysis results in a new CEX also transient FM1 lead to violation of SR whereas if no CEX is found a transient behaviour of FM1 has no impact on SR.

Another temporal aspect is to impose a "system state" under which the analysis shall be done. ISAAC platform makes it possible to define sequences of values for one or more of the design variables representing a system state. In the CEX, in

Figure 2, we may impose X1 and X2 to be constantly set to true and still allowing only FM1 and FM2 to become true. If SR is still violated it indicates that FM1 and FM2 is a cut set under the specified system state whereas if no CEX is found the occurrence of FM1 and FM2 under the specified system state has no impact on SR.

4. Specialized analysis based on formal behavioural models

In the following it will be shown how the approach for performing safety analysis in the "ESACS environment", described in the previous paragraphs, has been reused during the ISAAC project in order to cover other safety related aspects, especially considered during the design phase, like: the safety requirements allocation, the testability and the mission reliability analyses.

4.1 Requirement allocation based on SAP

When designing a system, engineers have to decompose high level system requirements into finer requirements that will constrain system subparts. Such allocation of requirement is indeed strongly connected to the definition of the system architecture. So, in order to assist the allocation of safety requirements, we proposed to encode experts' know-how into formal model libraries of typical safety micro-architectures [8].
The micro-architectures models exhibit elements of interest for a safer design: 1) structural features (e.g. redundancy), 2) good use condition and 3) induced safety properties. They are indeed abstract views of the system safety elements and will be called Safety Architecture Patterns (SAP).
Figure 3 gives two examples of patterns. On the right side, one can see a "testable basic block". The requirements allocated to this block are the following: it shall provide a health status "f" and an output "o" when it did not fail and it receives an input "i", an activation signal "a" and a resource "r". On the left side, one can see a cold spare redundancy mechanism made of two testable basic blocks C1 and C2 and activated by a controller that switches from C1 to C2.

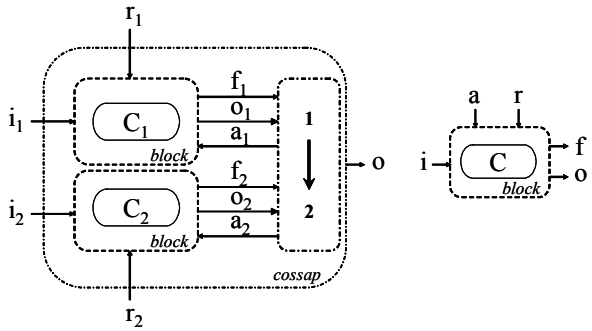

Figure 3 – *CoSSAP* and *Block*

Such patterns are often used to tolerate one failure (guaranteed property) under good use condition (derived requirements for interface system) such as:

```
Activation of C1 and no failure of C1
    always imply resource for C1
```

This derived requirement states that "when the primary component C1 is operational (because it is safe and activated), then it shall receive the resources that are necessary to provide the output".
In ISAAC, libraries of SAP are developed. Currently, the expected behaviours of a SAP are described within the AltaRica language whereas other associated requirements (good use conditions and guaranteed properties) are expressed by temporal logic formulae. Each SAP is pre-proved using the formal tools described previously, i.e. one verifies that under the expressed assumption of the SAP environment, the AltaRica model fulfils the guaranteed properties. Then the system model can be built as previously described. The novelty is that pre-defined safety requirements are made explicit and formal for further analysis.

4.2 Testability

With Testability we intend a design characteristic that allows the status (operable, inoperable, or degraded) of a system or an item to be determined and the isolation of faults within the system or the item to be performed in a timely manner.
The Testability characteristics of a complex system are very important also from a safety point of view, especially in the aeronautical field. Here Testability plays an essential role both in flight, to guarantee safety level (switching off or reconfiguring faulty items or systems) and condition awareness to the pilots (status indication), and on ground, during maintenance and repair procedures.

The idea is then to reuse the two basic and complementary strategies for the evaluation of the effects of failure events on a system model that is the inductive or "bottom-up" approach and the deductive or "top-down" approach, to verify testability related requirements.

In ISAAC we set as our goal the implementation of two different kinds of testability analysis and namely the *Fault Detection* and the *Fault Isolation* analysis.

The objective of Fault detection analysis is to find if and how faults are detected.
With "detected" we mean the issue of a particular signal (or set of signals) every time that the fault shows (that is when the failure mode is activated). The Fault Detection analysis, in the environment described in section 2 and 3, considers as top level event the generation of a detection signal and uses the bottom-up approach in order to highlight if and

how a given failure mode (or combination of failure modes) is detected in all the possible system's states.
A sub-case of Fault Detection Analysis is the False Alarm Analysis that consists in analysing if it is possible to activate a detection signal without the activation of any failure mode.

On the contrary the deductive approach (top-down approach) is followed in order to perform a Fault isolation analysis, that is, to see which combination of failure modes can lead to the activation of the given detection signal.
The result of this analysis is a "causal tree", that is, a tree that relates the issuing of a detection signal with the causal failure modes.
Starting from the "causal tree", one or more re-ordering strategies (based on, for instance, the failure rate, the time to re-test each failure mode and so on) could be used in order to identify the most appropriate fault isolation sequence.

4.3 Mission reliability analysis

Mission analysis' target is to determine the impact of degraded situations on the system operational modes and over pre-defined missions that define the scenarios in which the system being developed will be used.

Data representation
In case of mission analysis, not only the "system" has to be represented, but also the "mission".

The suggested model architecture is indicated in the following figure where two main "charts" are represented in parallel.

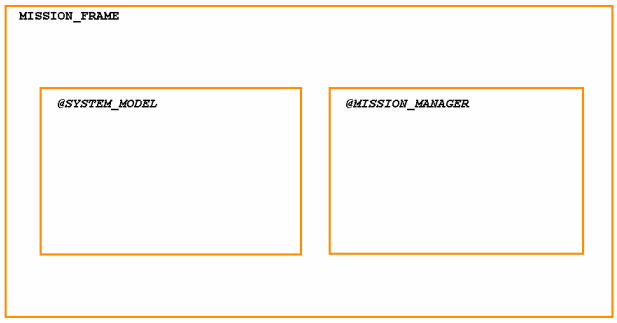

Figure 4: Model architecture for Mission Analysis

The first chart represents the system model.
A system that operates in a mission can be in different configurations depending on the mission phase. The "formal" model under this chart should therefore represent the system configurations and the transitions among them.

The second chart represents the mission model.

The mission has to be represented with its attributes: phases, transitions among them, duration and requirements.

An example of mission model, "Mission Manager", is indicated in the following figure.

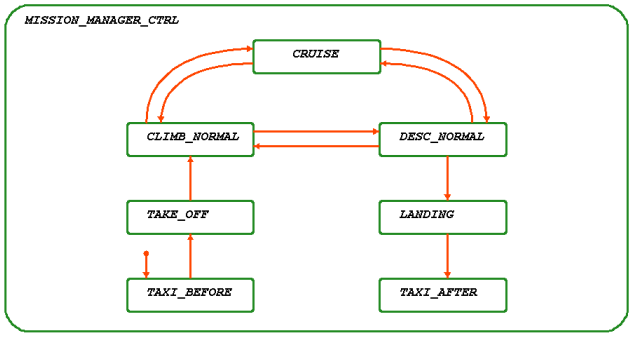

Figure 5 : Mission Manager structure

Qualitative analysis and results representation
The techniques described in paragraph 3 can be used to find failure mode sequences responsible for mission failure or leading to violation of specific mission requirements.
For the mission analysis purpose, the techniques are scaled up in order to include the possibility to define "observables" (e.g. mission phases) that are then used in the representation of the results.
See the following figure.

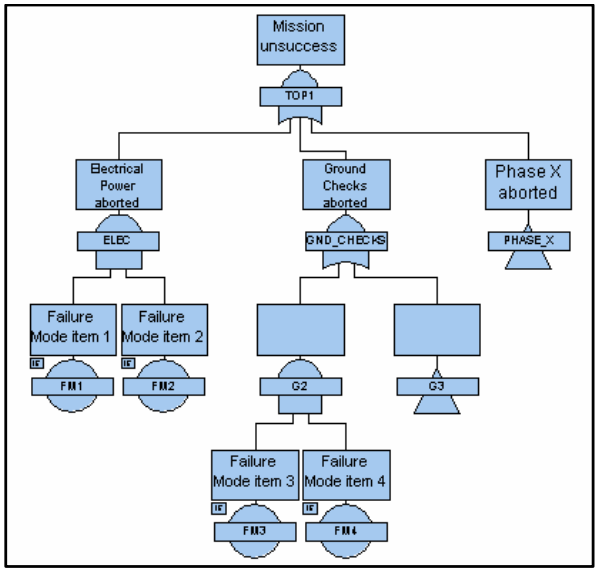

Figure 6: Results representation

Quantitative analysis
The failure modes sequences leading to the mission unsuccess with the relevant values for the failure rates can be used for performing the calculation of the Mission Reliability quantitative figure.

Other aspects
In an operational context other aspects are relevant: like the occurrence of a particular risk (e.g. an engine disk burst) or a pilot error.

Therefore a comprehensive environment for mission analysis integrates also the techniques that are described in the following paragraphs 5 and 6.

5. Coupling bevioural models of complex system and geometric models for particular risk analysis

5.1 Analysis of geometrical models

Geometric installation models of systems are created as the system is developed. Abstract blocks that serve as placeholders for components are initially introduced with subsequent refinements as more details regarding the components shape become available.

Particular risk models and trajectories are agreed with the authorities. The level of detail included in the models varies for each particular risk model.

3D modelling tools such as Catia or more specialised tools such as IRIS are used to cohabitate the systems installation geometry with the particular risk models and carry out a collision analysis.

The particular risk analysis type determines the parameters that are used to characterise each collision.

An example of a particular risk analysis model is uncontained engine rotor failure (UERF). Here the critical parameters are the Phi angle (roll angle) and the Khi angle (angle that the trajectory vector has with reference to the rotor disk plane). The resolution of the analysis depends on the distance from the fragment point of origin, the criticality of the effect and size of the accounted fragment. A failed rotor fragment from the Engine affects the components over a range of Phi and Khi angles. The risk window is then plotted for each component over the range. Defining the risk window for each component for the range of Khi and Phi angle within a sub-system and then overlaying them would give an idea of the level of redundancy the system has for each fragment.

The useful information from such an analysis includes the systems, structures and components that are simultaneously affected by each instance of a particular risk in addition to the energies that are involved; the spacing between the affected components, the probability of a trajectory being followed, etc. The relevant information used within the context of this project is related to the identification of groups of simultaneously affected components from all the systems that play a part in supplying the functionality that is stipulated by the safety requirement that is being analysed.

5.2 Coupling geometrical and behavioural worlds

We want to support the coupling of geometrical and behavioural models in both directions. From geometrical to functional world, we want to assess whether a trajectory computed with a particular risk tool is critical with respect to the system safety requirements. The set of items impacted by a trajectory is related with a failset i.e. a set of failures modes that are triggered simultaneously in the behavioural model. Once the behavioural model is extended with failsets, ESACS tools as described in section 3 can be used to check if the safety requirements are still valid.

We developed tools that define failsets based on particular risk results computed with tools as CATIA or IRIS and a table that relates geometrical and behavioural component names.

Work on the reverse direction (from functional to geometrical world) is under progress. Two capabilities will be developed: Visualisation of critical failsets into the 3D tools, Allocation to 3D components of requirements derived from the functional world analysis. These new capabilities could be used to guide the installation of equipments into the aircraft.

6. Coupling bevioural models of complex system and pilot models for human error analysis

The main objective of coupling behavioural models of complex systems and pilot models in ISAAC is to adopt the ESACS methodology to the requirements of an industrial Human Error Analysis (HEA). HEA in aerospace generally covers a multitude of areas from design, maintenance through to pilot behaviour. The general target of HEA in this project is to identify potential pilot errors and the safety impact on the overall flight.

Specific to ISAAC is the definition of a formal cognitive architecture of generic pilot behaviour. This architecture is capable to interpret/execute formal procedure models like takeoff or approach. Furthermore it is able to modify the procedure model based on a psychological plausible cognitive learning mechanism. These learned modifications may lead to pilot errors. Thus in the ISAAC-HEA the normative procedure models, also called mental models (MM), are extended by means of the learning mechanism, this leads to an extended mental models (EMM).

The focus is on mode errors, where an action is performed that is correct in some modes but not in the present one. Mode errors lead to "automation

surprises", where an operator no longer understands what the system is doing. During the design the need for modes has to be balanced against the probability of mode errors. Generally a mode may be understood as a system configuration with a specific functionality. Modern avionics systems are equipped with a huge number of different modes. This allows the use of systems in a variety of different operating condition, but at the same time it becomes difficult for the operators to retain “mode awareness”.

Lüdtke and Möbus [9] performed a simulator study with four pilots at the Lufthansa Flight Training Center. As a result they found that a subset of mode errors may be explained by “learned carelessness”. This psychological theory [10] states that humans have a tendency to neglect safety precautions if this has immediate advantages, e.g. it saves time. Careless behaviour emerges if safety precautions (like checking the actual mode before pressing an auto pilot button) have been followed several times but would not have been necessary, because no hazards occurred. Then, people deliberately omit safety precautions because they are considered a waste of time. The absence of hazardous consequences acts as a negative reinforcer of careless behaviour. Learned carelessness is a process which is characteristic for human nature because we have to simplify in order to be capable to perform efficiently in a complex environment. We implicitly degrade our mental model to optimise it for routine situations. Unfortunately this may be disastrous in slightly deviating scenarios. Thus it is crucial to consider this process in system design.

Learned carelessness was modelled inside a cognitive architecture based on the mechanism of rule composition and tested by comparing the model behaviour and real pilot behaviour in different flight procedures. These trials showed that the model commits errors that comply with errors observed in the empirical study [9].

Starting from these initial results the cognitive architecture and simulation platform are investigated in ISAAC with further more complex procedures, like arrival and takeoff. With regard to learned carelessness the specific analysis question will be answered, if during the interaction with the design under investigation in specified scenarios the pilot is likely to simplify his knowledge about certain flight procedures and if these simplifications may lead to pilot errors and violations of safety requirements. In ISAAC this question is tackled by simulation and verification. A simulation platform is developed that integrates the cognitive architecture, a system design and a flight simulation software (for the environment) and allows to simulate the dynamic interaction of these models with different procedures (e.g. takeoff or approach) and scenarios (e.g. standard profile takeoff and extreme weather and terrain takeoff).

A procedure has to be uploaded onto the cognitive architecture. At the beginning of the analysis this procedure contains only normative rules allowing to always reach the procedure goal successfully. After each simulation run, that means after a procedure has been simulated and the goal was reached, the pilot model simplifies the procedure model according to learned carelessness. This leads to an EMM with potentially hazardous procedure rules leading to pilot error. After the EMM has reached a stable state it is translated into the formal notation of the system model. This enables to perform a fault tree analysis with the techniques described in paragraph 3 in order to identify all possible scenarios where the new procedure rules lead to pilot errors with an impact on the overall safety of flight. The resulting fault tree contains human errors as basic events and violations of procedure goals (e.g. the aircraft does not reach and maintain the initial altitude after takeoff) as TLE.

In ISAAC extensions of the cognitive architecture will be investigated. The architecture is extendible, because of its modular structure and because it is based on an established cognitive core components, that have been applied by various researchers to model a number of different cognitive mechanisms.

An additional benefit of having mental models and system models represented in a uniform framework is the ability to investigate mismatches between the two. In the ISAAC context this exploited by the identification of “cognitive inadequate system structures”, i.e. system patterns that, although correct, are too difficult to operate on (e.g. because too much and/or too complex information needs to be kept in the mental model during operation). Thus, we cannot only investigate mental models as shown in [11], but are also able to reveal weaknesses in the interaction between the operator and the system in terms of system features.

7. Applicability of the approach

The system modelling approach and the first version of the generic safety assessment tools presented in sections 2 and 3 were successfully applied to several case studies during the ESACS project. Currently, new tools versions and the extension presented in section 4, 5 and 6 are under test in ISAAC project. The lessons learnt so far are the following.

The case studies are extracted from existing aircraft systems. Regarding, ESACS/ISAAC basic techniques, the sequences or cuts of failures

computed in ESACS/ISAAC projects seem to have at least the same accuracy than the pre-existing ones. This is due to two factors. First, the tools can deal automatically with numerous detailed failure modes. The issue is rather to find out which level of details is the most appropriated with respect to the analysis purpose. Second, the tools can also provide more details about the temporal order between events that lead to a top level event.

Moreover, the approach really eases the dialog between safety engineers and system designers. Consider the possibility to show sequences of failures directly in a simulation model. This would help, in respect of usually huge Fault Trees, with the understanding of the system behaviour, providing evident support that the design satisfies the safety objectives.

It is worth noting also that the set of case studies is a significant sample of safety critical embedded system. It includes not only command systems that control aircraft mechanical components (e.g. flight control or landing gear systems) but also systems that provide resources for the others (e.g. hydraulic and electrical power generation and distribution). Thus the method seams to be applicable for a wide range of systems embedded not only in aircrafts but also on board of cars, trains or space vehicles.

Regarding newer experiments of ISAAC, they may testify of a wider application range. For instance, mission reliability analyse or the human error analysis require to deal with human models. Today, only results of preliminary tests are available. They shown the feasability of the new concepts; further results are expected to discuss the concept maturity and applicability in an industrial context.

Last but not least, the project partners promote the new methods towards the Authorities as applicable means of compliance for Certification purposes. For civil aircraft, the Certification process, as indicated in the Aerospace Recommended Practice - ARP 4754 “Certification Considerations for Highly Integrated or Complex Aircraft Systems”, requires evidence that the safety requirements and objectives are satisfied by means of safety assessment analyses that are performed during all system development cycles. Usually Fault Trees, FMEAS… constitute the main elements for this evidence. ISAAC proposes formal models and the associated tools as additional means of compliance; these means are not referred today in the ARP documents.

Nevertheless, several general presentations of ESACS/ISAAC were already performed with good feedback from representatives of the certification authorities or in the SAE group in charge of updating the relevant ARP documents.

Moreover, Dassault Aviation chose to certify the flight by wire system of the Falcon 7x using the system model based approach. New steps in the certification process were defined and agreed both by European and American certification authorities.

This is a first acceptance step. The next step is to register the underlying methodology in a forthcoming update of the ARP 4754/4761 document guide. DASSAULT AVIATION and AIRBUS will jointly support this activity in the next SAE comity session.

8. Conclusion

This paper has presented a safety assessment methodology based on formal modelling and reasoning methods. The methodology consists of two alternative approaches. First, the analysis can be performed directly on system design models (e.g. SCADE, Statemate or Simulink models) and merely extended, by analysts, by the injection of failure behaviour into model variables. Alternatively, the analysis can be based on formal safety models constructed by analysts; such models (normally expressed in AltaRica language) abstract from the “nominal” behaviour of the system and, instead, explicitly capture system behaviour in presence of failures. Both approaches allow automating traditional safety analyses (e.g. FTA). They also allow users (analysts) to undertake numerous “what if” investigations, for example to investigate whether the effect of a particular cut set is dependent on the order and/or duration of failure modes in the set.

This paper has also described extensions to the baseline methodology that are being addressed in the ongoing ISAAC project.

There are two types of extensions. The first type further utilises the capabilities of model checkers and exploits the information contained in the models in order to deliver new types of analyses. This includes: assistance in allocation of derived safety requirements based on formal safety models and libraries of pre-defined “building blocks” – safety architecture patterns, mission reliability analysis that utilises the structure of the mission tree in order to provide more accurate predictions on reliability characteristics of the system and analyses concerned with the coverage achieved by health monitors and properties of such monitors (testability and detectability analyses). The purpose of this type of extensions is to maximise applicability of the ISAAC methodology at different stages of the design and assessment processes of safety critical systems, starting from early design, through to the later detailed stages of the PSSA.

The goal of the second type of methodology extensions studied by the ISAAC project is to extend the methodology to include different types of

analyses in order to achieve better integration with what is often perceived as relatively independent and auxiliary analyses (e.g. CCA, PRA, etc) and in order to take advantage of the benefits gained from the use of formal methods in such analyses. Therefore, these extensions focus on increasing the scope of the models being analysed, or, to be more precise, by linking "core" behavioural models with both geometrical and cognitive models to allow analysis to cover the effects of common causes (due to the layout of equipment) and human errors (e.g. due to particular training procedures and typical operation profiles) respectively.

Finally, the ISAAC project is undertaking a number of purely methodological investigations in order to prepare theoretical grounds for yet further extensions. This includes a possibility to further utilise and extend capabilities of tools to analyse temporal aspects of the system behaviour and, thus, to provide analysts with a more detailed and precise description of how individual failure modes propagate and combine in the system eventually leading to unsafe conditions or behaviour. The investigations cover not only extensions to the tools but also extending traditional combinatorial analysis methods (e.g. FTA).

Both ESACS and ISAAC projects have included a variety of industrial partners – all leaders in European aeronautics market. Consequentially, the methodology has been applied to real industrial case studies and the results from the evaluation were quite positive and encouraging.

Furthermore, one of the industrial partners has successfully negotiated with the European Aviation Safety Authority precise conditions for using the methodology to produce evidence for aircraft certification.

To conclude, it is important to note that although the project has focussed on aeronautics domain, nothing in the safety assessment methodology presented here cannot be successfully reused in any other industrial context that is concerned with safety critical systems, safety engineering and assurance.

9. Acknowledgement

- The European Union that sponsored and funded ESACS and ISAAC projects

- Several other people contributed to the work presented in this paper. We wish in particular to thank: M. Delpiano and M. Fabbri from Alenia Aeronautica, M. Mahrun, M. Fortes Da Cruz, M. Frisk, L. Goutal, D. Javeaux, C. Jourdan, F. Rezeaux, from Airbus, J. Karlsson and P. Persson from SAAB, L. Borgne from Prover, A. Sboner from IRST. We would also like to thank B. Lawrence and R. Knepper from Airbus for helping us improve this paper.

10. References

1. M. Bozzano, A. Villafiorita, O. Åkerlund, P. Bieber, C. Bougnol, E. Böde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi, A. Cimatti, A. Griffault, C. Kehren, B. Lawrence, A. Lüdtke, S. Metge, C. Papadopoulos, R. Passarello, T. Peikenkamp, P. Persson, C. Seguin, L. Trotta, L. Valacca, G. Zacco *"ESACS: an integrated methodology for design and safety analysis of complex systems"*, in proceedings of ESREL 2003, Balkema publisher.
2. Society of Automotive Engineers Inc, Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems, November 1996.
3. Society of Automotive Engineers Inc, Aerospace Recommended Practice (ARP) 4761: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, December 1996.
4. A. Arnold, A. Griffault, G. Point and A. Rauzy. "*The AltaRica formalism for describing concurrent systems".* Fundamenta Informaticae, 40(2-3): 109-124, 1999.
5. C. Kehren, C. Seguin, P. Bieber; C. Castel, C. Bougnol, J.-P. Heckmann and S. Metge. *Advanced Multi-System Simulation Capabilities with AltaRica*, proceedings of Safety Critial System Conference, Rhodes Island, USA, august 2004.
6. T. Peikenkamp, E. Böde, I. Brückner, H. Spenke, M. Bretschneider, and H.-J. Holberg. Model-based Safety Analysis of a Flap Control System. In Proceedings of the INCOSE 2004 -- 14th Annual International Symposium, Toulouse, 2004.
7. Huth, Michael R.A. and Ryan, Mark D.: Logic in Computer Science – Modelling and reasoning about systems, Cambridge University Press, 2001
8. C. Kehren, C. Seguin, P. Bieber, C. Castel, C. Bougnol, J.-P. Heckmann, S. Metge, "Architecture Patterns for Safe Design", AAAF 1st Complex and Safe Systems Engineering Conference (CS2E 2004) , 21-22 juin 2004, Arcachon (France)
9. Lüdtke, A., Möbus, C.: *A Cognitive Pilot Model to Predict Learned Carelessness for System Design.* In A. Pritchett and A. Jackson (Ed.). Proceedings of the International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero), 29.09.-01.10.2004, Toulouse, France. CD-ROM, 2004
10. Frey, D., Schulz-Hardt, S.: Eine Theorie der gelernten Sorglosigkeit. In H. Mandl (Hrsg.), 40. Kongress der Deutschen Gesellschaft für Psychologie, 604-611. Göttingen, Seattle: Hogrefe Verlag für Psychologie, 1997
11. Rushby, J.: Using model checking to help discover mode confusions and other automation surprises, in 'Proceedings of the Workshop on Human Error, Safety, and System Development (HESSD) 1999', Liège, Belgium

11. Glossary

CCA: Common Cause Analysis
CEX: Counter Example
COSSAP Cold Spare Safety Architecture Pattern
EMM: Extended Mental Model
ESM: Extended System Model
FM: Failure Mode
FMEA: Failure Mode and Effect Analysis
FTA: Fault Tree Analysis
MM: Mental Model
PRA: Particular Risk Analysis
PSSA: Preliminary System Safety Assessment
SAP: Safety Architecture Pattern
SM: System Model
SR: Safety Requirement