Formal Design and Safety Analysis of AIR6110 Wheel Brake System

M. Bozzano[1], A. Cimatti[1], A. Fernandes Pires[1], D. Jones[2], G. Kimberly[2], T. Petri[2], R. Robinson[2], and S. Tonetta[1]

[1]Fondazione Bruno Kessler, Trento, Italy
[2] The Boeing Company, P.O. Box 3707, Seattle, WA 98124, USA

Abstract. SAE Aerospace Information Report 6110, "Contiguous Aircraft/System Development Process Example," follows the development of a complex wheel brake system (WBS) using processes in the industry standards ARP4754A, "Guidelines for Development of Civil Aircraft and Systems," and ARP4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment."
AIR6110 employs informal methods to examine several WBS architectures which meet the same requirements with different degrees of reliability.
In this case study, we analyze the AIR6110 with formal methods. First, WBS architectures in AIR6110 formerly using informal steps are recreated in a formal manner. Second, methods to automatically analyze and compare the behaviors of various architectures with additional, complementary information not included in the AIR6110 are presented. Third, we provide an assessment of distinct formal methods ranging from contract-based design, to model checking, to model based safety analysis.

Keywords: Aerospace Recommended Practices, Case Study, Model Checking, Safety analysis, Fault Tree, Contract-based design

1 Introduction

General Context As aerospace systems become more complex and integrated, it becomes increasingly important that the development of these systems proceeds in a way that minimizes development errors. Advisory Circular (AC) 20-174 [13] from the FAA specifies the Society for Automotive Engineering (SAE) guidance, Aerospace Recommended Practice (ARP) ARP4754A [24], "Guidelines for Development of Civil Aircraft and Systems," as a method (but not the only method) for developing complex systems. ARP4754A along with its companion ARP4761 [23], "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," provide the guidance that original equipment manufacturers (OEMs) such as Boeing and Airbus may utilize to demonstrate that adequate development and safety practices were followed, and that final products meet performance and safety requirements while minimizing development errors.

System safety assessment is a development process compatible with ARP4761 which ensures that system architectures meet functional and safety requirements. Architecture decisions take system functions and safety into account through the use of

countermeasures to faults such as redundancy schemas, fault reporting, maintenance, and dynamic system reconfiguration based on fault detection, isolation, and recovery (FDIR). The role of safety assessment is to evaluate whether a selected design is sufficiently robust with respect to the criticality of the function and the probability of fault occurrence. For example, functions with catastrophic hazards must not have any single failure that can result in that hazard. Also, each level of hazard category (catastrophic, hazardous, major, minor) has an associated maximum probability that must be ensured by the design. For all functions, the system architecture and design must support availability and integrity requirements commensurate with the functional hazards. Among the various analyses, the construction of fault trees [26] is an important practice to compare different architectural solutions and ensure a compliant design.

The AIR6110 *document* Aerospace Information Report (AIR) AIR6110 [25] is an informational document issued by the SAE that provides an example of the application of the ARP4754A and ARP4761 processes to a specific aircraft function and implementing systems. The non-proprietary example of a wheel brake system (WBS) in this AIR demonstrates the applicability of design and safety techniques to a specific architecture. The WBS in this example comprises a complex hydraulic plant controlled by a redundant computer system with different operation modes and two landing gears each with four wheels. The WBS provides symmetrical and asymmetrical braking and anti-skid capabilities. AIR6110 steps the reader through a manual process leading to the creation of several architectural variants satisfying both functional and safety requirements, and cost constraints.

Contribution In this paper, the informal process employed in AIR6110 is examined and enhanced using a thorough, formal methodology. We show how formal methods can be applied to model and analyze the case study presented in AIR6110. This formal method supports multiple phases of the process, explores the different architectural solutions, and compares them based on automatically produced artifacts.

The formal modeling and analysis are based on the integration of several techniques, supported by a contract-based design tool (OCRA [7]), a model checker (NUXMV [6]), and a platform for model-based safety analysis (XSAP [1]). Using these techniques and tools, we create models for the various architectures described in AIR6110, demonstrate their functional correctness, and analyze a number of requirements from the safety standpoint, automatically producing fault trees with a large number of fault configurations, and probabilistic reliability measures.

Distinguishing features The work described in this paper is important for several reasons. First, it describes a fully-automated analysis of a complex case study, covering not only functional verification but also safety assessments. Second, we propose the integration of different formal techniques (e.g., architectural decomposition, contract-based design, model checking, model-based safety analysis, and contract-based safety analysis) within an automated, unifying flow, which we analyze in terms of scalability and accuracy. Finally, we report interesting results from the standpoint of the AIR6110. Specifically, we provide qualitative and quantitative analyses of the WBS, through an

examination of the respective merits of the various architectures. We also show that a flaw affects more architectures than reported in AIR6110.

Related work The WBS described in ARP4761 has been used in the past as a case study for techniques on formal verification, contract-based design and/or safety analyses (see, e.g., [20, 21, 11, 9]). With respect to these works, this case study is much more comprehensive, describes a more elaborate design, and is the only one to automatically produce fault trees. In [3], contract-based fault-tree generation is applied to the ARP4761 WBS, but on a much smaller architecture than those considered in this paper. Moreover, the current work is unique in the literature, in that it takes into account the process described in AIR6110 and analyzes the differences between the various architectures.

There are many applications of formal methods in the industrial avionics process. The ESACS [12], ISAAC [19], and MISSA [22] projects pioneered the ideas of model extension and model-based safety assessment, and proposed automatic generation of fault trees. However, we are not aware of other significant case studies combining contract-based design, formal verification, and model-based safety analysis (with automated fault tree generation) as in the methodology described in this paper.

Plan of the paper. In Section 2 we present the informal AIR6110 application and process. In Section 3 we give an overview of our formal process. In Section 4 we discuss the formal models of the WBS. In Section 5 we present the results of the formal analyses. In Section 6 we discuss the lessons learned, and outline future work.

2 AIR 6110

2.1 Overview of the standards

ARP4754A [24] and ARP4761 [23] define Recommended Practices for development and safety assessment processes for the avionics field. The practices prescribed by these documents are recognized by the Federal Aviation Administration (FAA) as acceptable means for showing compliance with federal regulations [13, 14], and have been used by the industry of the field for years.

The Aerospace Information Report 6110 (AIR6110) document was released by SAE in 2011. It describes the development of sub-systems for a hypothetical aircraft following the principles defined in ARP4754A, and shows the relationships with the ARP4761. AIR6110 focuses on the Wheel Brake System (WBS) of a passenger aircraft designated model S18. The hypothetical S18 aircraft is capable of transporting between 300 and 350 passengers, and has an average flight duration of 5 hours.

2.2 Overview of the Wheel Brake System

The WBS of the S18 is a hydraulic brake system implementing the aircraft function *"provide primary stopping force"*. In particular, it provides braking of the left and right main landing gears, each with four wheels. In addition to coupled braking, each landing gear can be individually controlled by the pilot through a dedicated (left/right) brake pedal.

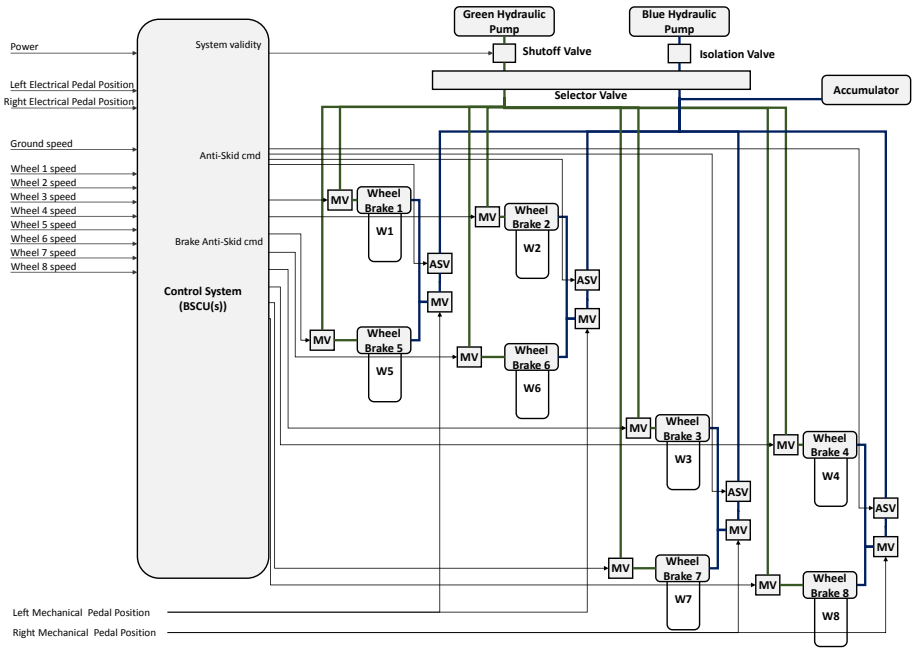

Fig. 1. WBS architecture overview (MV=Meter Valve ; ASV=AntiskidShutoff Valve ; W=Wheel)

WBS architecture and behavior An overview of the WBS architecture is shown in Figure 1. For the sake of clarity, the control system is not decomposed and sensors for the pedal position and the wheels' angular speed are not represented.

The WBS is composed of a physical system and a control system. The physical system includes hydraulic circuits running from hydraulic pumps to wheel brakes and thus providing braking force to each of the 8 wheels. The physical system can be electrically controlled by the Braking System Control Unit (BSCU) of the control system, or mechanically controlled directly through the pedals' mechanical position, depending on the operation mode of the WBS.

There are 3 different operation modes. In *normal mode*, braking is effected by the primary hydraulic circuit, referred to as the green circuit. The green circuit is composed of a hydraulic pump and 8 meter valves, one valve for each wheel. Each valve is individually controlled by electrical commands provided by the BSCU. The BSCU signals a combined brake and antiskid command, which may be different for each wheel. The brake command depends on the electrical signal received from the pilot pedal. The antiskid command is computed based using sensor inputs that indicate ground speed, wheel speed and brake command.

In *alternate mode*, braking is effected by a second hydraulic circuit, called the blue circuit. The 8 wheels are mechanically braked in pair, 2 pairs per landing gear. The blue circuit is composed of a hydraulic pump, 4 meter valves and 4 anti-skid shutoff valves. Each meter valve is mechanically commanded by its associated pilot pedal. The

switch between green and blue circuits is mechanically controlled via a selector valve. When the selector valve detects a lack of pressure in the green circuit, it automatically switches to the blue circuit. When the green circuit becomes available again, it switches from the blue circuit back to the green circuit. A lack of pressure in the green circuit can occur if the hydraulic pump of the circuit fails or if the pressure is cut by a shutoff valve. The shutoff valve is closed if the BSCU becomes invalid.

Emergency mode is supported by the blue circuit, operating only in case the hydraulic pump fails. In this case, an accumulator backs up the circuit with hydraulic pressure, supplying sufficient pressure to mechanically brake the aircraft. An isolation valve placed before the pump prevents pressure from flowing back to the pump.

WBS requirements The AIR6110 document contains several requirements for the WBS. These can be grouped in two main categories: Requirements corresponding to safety, e.g., *the loss of all wheel braking shall be extremely remote*, and others, e.g., *the WBS shall have at least two hydraulic pressure sources*.

Our case study focuses on five safety requirements, that are well representative of safety requirements that should be handled during safety assessment:

S18-WBS-R-0321 *Loss of all wheel braking (unannunciated or annunciated) during landing or RTO shall be extremely remote*

S18-WBS-R-0322 *Asymmetrical loss of wheel braking coupled with loss of rudder or nose wheel steering during landing or RTO shall be extremely remote*

S18-WBS-0323 *Inadvertent wheel braking with all wheels locked during takeoff roll before V1 shall be extremely remote*

S18-WBS-R-0324 *Inadvertent wheel braking of all wheels during takeoff roll after V1 shall be extremely improbable*

S18-WBS-R-0325 *Undetected inadvertent wheel braking on one wheel w/o locking during takeoff shall be extremely improbable*

Intuitively, a safety requirement associates the description of an undesirable behaviour or condition (e.g. “inadvertent wheel braking”) with a lower bound on its likelihood, according to terminology (e.g. “extremely improbable”) defined in [15].

2.3 The informal development process

The AIR6110 document describes the development process shown in Figure 2 as applied to the WBS. It details the development of the WBS system architecture in four versions, each obtained after design choices of different types.

ARCH1 is the high-level architecture of the WBS. It represents the first step in the architecture definition by defining the main functional elements of the WBS. It incorporates only one hydraulic circuit and one Control Unit.

ARCH2 is the first concrete architecture which meets basic safety requirements. The development of ARCH2 is motivated by performing a Preliminary System Safety Assessment (PSSA) on ARCH1, which results in the introduction of redundancy in the hydraulic circuits and in command computation with two BSCUs in the control system.

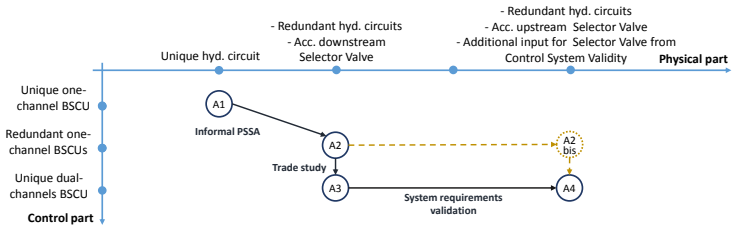

Fig. 2. Five architectures

ARCH3 is motivated by trade studies on ARCH2. The purpose of the trade study is to assess other architectures answering to the same safety requirements as ARCH2 and which are less expensive and easier to maintain. Only the control system architecture is modified by going from two BSCUs to one dual-channeled BSCU.

ARCH4 is driven by validation of safety requirements on ARCH3. Specifically, a safety requirement addressing mutual exclusion of the operating modes of the WBS is shown to be unmet. ARCH3 is modified to meet the requirement. Only the physical system is modified, by adding an input to the selector valve corresponding to the validity of the control system and moving the accumulator in front of the selector valve.

For our case study, we added an architecture variant called ARCH2BIS which is based on the control system architecture of ARCH2 and the physical system architecture of ARCH4. The purpose is to show that it is possible to detect the issue that motivated the change to ARCH4 earlier in the design process at ARCH2.

3 Formal approach

The formal approach to modeling and analyzing the case study follows the process depicted in Figure 3. The main steps are: component-based modeling of the system architecture and contract-based specification of the architectural decomposition; definition of the behavioral implementation of components at the leaves of the architecture, generation of the full system implementation and formal verification of the properties; extension of the model with failures to include faulty behaviors of components; production of a safety analysis based on fault-tree analysis.

The formal approach is supported by a set of tools developed by the Fondazione Bruno Kessler, namely OCRA [7, 17, 9, 10] for contract-based specification, verification, and safety analysis of the architecture decomposition; NUXMV [6, 16] for formal verification of the behavioral implementation; and XSAP [1, 18, 4] for model-based safety analysis of the behavioral implementation.

Formal verification of the architectural decomposition The architecture decomposition is expressed in the OCRA language and the component contracts are expressed in linear temporal logic (LTL). The system architecture is hierarchically decomposed into constituent components, until leaf components of the system are reached. Each component has an interface defining the boundary between the component implementation and

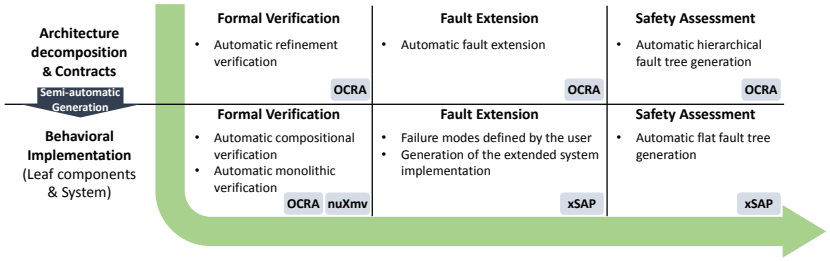

Fig. 3. Overview of the process and the related tool support.

its environment. An interface consists of a set of input and output ports through which the component implementation interacts with its environment. A composite component is refined into a synchronous composition of sub-components. The decomposition also defines interconnections among the ports of the subcomponents and the composite component. The implementation of a composite component is given by the composition of the implementations of the subcomponents. Similarly, the environment of a subcomponent is given by the composition of the other subcomponents.

The properties in component contracts are formalized into LTL formulas following the Contract-Based Design supported by OCRA. Each component is enriched with contracts that define the correct refinement of the architecture. A contract is composed of an `assume` clause, which represents the property that the environment of the component must ensure, and a `guarantee` clause which describes the property that the component must ensure. Contracts of refined components are refined by the contracts of their sub-components. This refinement can be automatically checked by OCRA by generating and discharging a set of proof obligations that are validity problems for LTL.

Formal verification of the behavioral implementation OCRA can also generate implementation templates for leaf components of the architecture. Implementation templates are generated in the SMV language [6]. The user needs to provide only the implementations of leaf components in the template. Once done, OCRA can take into account these implementations to automatically generate a full system implementation in the SMV language. During this generation, the component contracts are automatically translated as LTL properties in the system implementation. Each leaf component implementation can be checked according to the component contracts defined in the architecture decomposition using OCRA. The full system implementation can also be monolithically checked using the symbolic model checker NUXMV.

Model-Based Safety Analysis (MBSA) XSAP is used to support extending a nominal model with failure modes provided by the user. A failure mode represents the behavior of a component in the context of a given failure. Failure modes can be defined from the XSAP fault library using a dedicated language for fault-extension. Once failure modes are defined for each component, XSAP can proceed to the fault extension of the nominal model and generate a new SMV implementation taking into account failure behaviors.

This extended model is used to conduct Model-Based Safety Analysis on the system using XSAP. More precisely, XSAP can generate flat fault trees based on this extended model. Such fault tree is a set of Minimal Cut Sets (MCS), which are the minimal configurations of faults leading to a Top Level Event (TLE). Here, the TLEs of the fault trees are the violations of the LTL properties resulting from the contracts provided in the architecture decomposition. Notice that a probability can be attached to each failure mode by the user, which will allow XSAP to compute the probability for the TLE to happen.

Contract-Based Safety Analysis (CBSA) An alternative way to perform safety analysis is provided by OCRA given the contract-based specification of the architecture [3]. The architecture decomposition is automatically extended with failure modes based on the contracts to generate a hierarchical fault tree. The TLEs of these fault trees are violations of the system contracts of the architecture. The intermediate events are violations of the subcomponents' contracts.

4 Formal models

4.1 Modeling nominal aspect

Key features The WBS architectures presented in AIR6110 are modeled following the formal approach described in Section 3. In order to formalize the case study, we applied some simplifying abstractions to the concrete system. First, we consider the hydraulic circuits as a unidirectional circuit, thus avoiding relational modeling of the circuit. As a consequence, the isolation valve present in Figure 1 is not relevant for the modeling and is removed from our models.

Another abstraction concerns the representation of hydraulic pressure in the hydraulic components, for example at the valve interfaces. All ports representing hydraulic pressure are expressed as bounded integers between 0 and 10 (represented as enumeration), as are ports representing braking force. A similar abstraction is applied to commands sent by the BSCU. All commands are represented as boolean values. The angular speed of each wheel is treated similarly. The angular speed of a wheel is represented by a wheel status, stopped or rolling. Under this representation, the wheel is considered to be skidding if the aircraft is moving and the wheel is stopped. These choices were made to limit complexity of the models while keeping a sufficient level of detail to obtain relevant results from the analysis.

We consider two behaviors for pressure supplied to hydraulic circuits. First, a hydraulic pump supplies hydraulic pressure only if the pump is supplied by electrical power and hydraulic fluid. This allows emphasizing the different mode changes defined in the WBS, depending on the pressure supply of each circuit. Second, the accumulator is considered to have an infinite reserve of pressure. This choice is justified by the fact that the model does not incorporate a concept of measuring "sufficient" pressure necessary to brake the aircraft.

Finally, all models are defined using discrete time and all component behaviors are instantaneous, i.e., all inputs are computed at the same time step where inputs are

provided. There is only one exception that concerns the wheel component: Braking force applied on the wheel determines the status of the wheel at the next step.

Architecture decomposition The decomposition of the WBS architectures is accomplished according to information provided in the AIR6110, extended by clarifications from brake system subject matter experts. Each architecture is decomposed into numerous sub-components. For example, the BSCU is decomposed into sub-modules that monitor the system and that create commands. The wheel brake is decomposed into a hydraulic fuse[1], a hydraulic piston and a brake actuator. Metrics for the different architectures are given in Table 1.

Behavior implementation The implementation of the leaf components is provided by the user, based on the implementation templates generated by OCRA. The architecture decomposition allows a wide reuse of the leaf component implementations through the architecture variants. For example, the leaf component implementations of ARCH2 and ARCH3 are identical. The only differences are due to the architecture decomposition at upper-levels, as introduced in Section 2. Similarly, ARCH2BIS and ARCH4 have the same leaf components; they differ from ARCH2 and ARCH3 in the implementation of the selector valve that is not only commanded by the pressure in inputs, but also by the control system validity. The largest delta between architectures is the change from ARCH1 to the rest. Due to the lack of redundancy, some of the leaf component implementations are different from ARCH2, ARCH2BIS, ARCH3 and ARCH4, or even not present at all, e.g., the selector valve.

Table 1. Formal modeling metrics

			ARCH1	ARCH2	ARCH2BIS	ARCH3	ARCH4
Architecture decomposition	Total components types		22	29	29	30	30
	Leaf components types		15	20	20	20	20
	Total components instances		100	168	168	169	169
	Leaf components instances		79	143	143	143	143
	Max depth		5	5	5	6	6
	Nb. contracts		121	129	129	142	142
System Implementation	Nb. properties		199	291	291	304	304
	State variables	Bool.	31	79	79	79	79
		Enum	55	88	88	88	88
Extended System Implementation	Failure modes		28	33	33	33	33
	Fault variables		170	261	261	261	261
	State variables	Bool.	74	156	156	156	156
		Enum	184	311	311	311	311

The full system implementation is automatically generated by OCRA, using the architecture decomposition and the leaf component implementations. The translation

[1] We consider that the hydraulic fuse is a simple pipe in the nominal model.

from the OCRA architecture to the SMV file preserves the structure, leveraging the use of modules. The contracts present in the OCRA file are translated as LTL properties in the system implementation. Metrics about the system implementation are given in Table 1, where we report the number of state variables and the number of property instances available for the system implementation, based on the properties generated from the contracts for each component type.

Requirements formalization and decomposition The five safety requirements expressed in Section 2.2 are translated as contracts at the system level. They are modeled as follows: First, we remove flight phase and speed value from the requirements, as we do not have sufficiently details about them in the models. The treatment of the required likelihood is ignored in the modeling, and is delayed to the phase of safety analysis. The undesirable condition is instead stated never to occur. For example, the requirements S18-WBS-0323 becomes "never inadvertent wheel braking with all wheels locked".

```
COMPONENT MeterValve
 INTERFACE
  INPUT PORT elec_cmd: boolean;
  INPUT PORT mech_cmd: boolean;
  INPUT PORT hyd_pressure_in: 0..10;
  OUTPUT PORT hyd_pressure_out: 0..10;

  CONTRACT apply_command
  assume: true;
  guarantee: always(((elec_cmd or mech_cmd) and hyd_pressure_in>0)
                    iff (hyd_pressure_out>0));
```

Listing 1.1. Architectural specification in OCRA for the Meter Valve

In addition, the requirements S18-WBS-R-0322 is split into two different contracts, one for the left side and one for the right side. The requirement S18-WBS-R-0325 is also split in eight contracts, one for each wheel.

In the subsequent phase of contract decomposition, these five safety requirements are in turn broken down into contracts for sub-components, describing the properties they must ensure, based on the description provided in AIR6110 and clarifications provided by subject matter experts. Additional contracts are also added to ensure the expected behavior of each component (e.g., braking force is applied when commanded). The number of contracts defined on each architecture is given in Table 1.

These contracts are then automatically translated into LTL properties in the system implementation, as described in Section 3.

```
MODULE MeterValve(elec_cmd, mech_cmd, hyd_pressure_in)
 VAR
   hyd_pressure_out : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
 LTLSPEC NAME apply_command_norm_guarantee :=
  (TRUE -> G(((elec_cmd | mech_cmd) & hyd_pressure_in>0) <-> hyd_pressure_out>0));
 ASSIGN
   hyd_pressure_out :=  ((mech_cmd | elec_cmd) ? hyd_pressure_in : 0) ;
```

Listing 1.2. Example of SMV implementation for the Meter Valve

Example of the Meter Valve A meter valve is a valve that will open if it receives a command. There are two possible commands: electrical or mechanical, both described

as boolean in our model. The specification of the Meter Valve in the architecture decomposition is given Listing 1.1. Its behavioral implementation is given Listing 1.2.

In this implementation, the only part added by the user is the `ASSIGN` part. All the rest is automatically generated by OCRA based on the architecture decomposition.

4.2 Modeling safety aspects

Failure modes and extended model A set of failure modes for the WBS has been defined for each component, based on expert clarifications: hydraulic pumps can fail to supply pressure to the hydraulic circuit; valves can fail open or closed (meter valves and antiskid shutoff valves can also be stuck at the last commanded position or at a random position); the accumulator can fail open or closed; hydraulic fuses can close the circuit; brake pistons and brake actuators can be stuck on, or off, or at the last position; wheels can fail with no rotation; sensors or BSCU components can send erroneous signals: the selector valve (occurring in ARCH2, ARCH2BIS, ARCH3, ARCH4) can fail only to its last position; switch gates (occurring in the control system of ARCH2, ARCH2BIS, ARCH3, ARCH4) can fail at the last position or in an intermediate position.

All failure modes described above can be encoded using predefined failure modes in the XSAP fault library. The user specifies possible failure modes for each component (a single component may be associated to more than one failure mode) using XSAP fault extension language. Then, the generation of the extended model is automatically carried out by XSAP. The extended model is typically two to three times larger than the model before extension.

An example of failure mode for the meter valve is given Listing 1.3. It describes a failure "failed closed" based on the predefined failure mode *StuckAtByValue* available in the XSAP fault library. For more details about the fault extension language, see [18].

```
EXTENSION OF MODULE MeterValve
   SLICE MeterValve_faults AFFECTS hyd_pressure_out WITH
     MODE MeterValve_FailedClosed {3.25e-6} : Permanent StuckAtByValue_I(
              data term << 0,
              data input << hyd_pressure_out,
              data varout >> hyd_pressure_out,
              event failure);
```

Listing 1.3. Example of failure mode "failed closed" for the Meter Valve

Availability of the fault extension affords a number of advantages. First, it guarantees the alignment of the nominal and extended models, avoiding a typical problem in safety analysis practice. Second, it saves a substantial amount of tedious modeling of faults, and thus improves productivity. Third, as for the implementation of the leaf components, the definition of failure modes for each component allows a similar rate of reuse for each architecture. Metrics about the failure modes and the extended model of each architecture are given in Table 1. To conduct a safety analysis on the extended models, the violation of the safety requirements described above are used as TLE.

Contract Based Safety Analysis (CBSA) The fault extension for the CBSA is automatically managed by the tool on the architecture decomposition. In comparison with the fault extension provided by the user in the MBSA, the CBSA approach takes into account all possible failure modes that will disprove the contract of a component.

5 Automated analyses

For the models described in the previous section, we carried out an experimental evaluation along the following dimensions: formal verification, construction of Fault Trees, and comparison of architectures.

Formal Verification The monolithic models, in form of SMV files, were analyzed with respect to properties resulting from the contracts in the architecture. We used NUXMV, running different verification engines: BDD-based model checking and IC3 [5]. The same results were also obtained via the contract-based approach of OCRA. The contract based verification process is based on the following steps: the top level properties are stated as contracts in the form of temporal logic formulae at the system level; each component is associated with contracts; the correctness of each contract refinement is proved by means of temporal entailment checks; the SMV module associated with each leaf component is proved to correctly implement the corresponding contracts.

All experiments have been performed on a cluster of 64-bit Linux machines with 2.7 Ghz Intel Xeon X5650 CPU, using one core with a memory limit set to 10Gb. The results are reported in Table 2 (with Ref. and Impl. representing time taken for the contract refinement and leaf implementation checks).

The results show that the compositional approach is often faster than the monolithic analysis. Consider also that contract refinement and implementation checks are fully independent and localized, and can in principle be executed in parallel. The VPar column in Table 2 reports maximum computation time across various individual checks, corresponding to the limit case where each check is run on a dedicated machine.

Table 2. Results of the Formal Verification (time in seconds)

	Monolithic approach		Compositional approach			
Arch	BDD	IC3	Ref.	Impl.	Tot.	VPar
ARCH1	38.32	56.62	1422.24	6.07	1428.31	439.62
ARCH2	2700.64	153.28	102.04	1.26	103.30	24.12
ARCH2BIS	3069.82	153.19	32.38	1.26	33.64	1.39
ARCH3	2935.88	159.01	72.87	1.29	74.16	10.74
ARCH4	3429.59	158.51	29.74	1.29	31.03	1.78

Aside from performance considerations, the most important result of the formal verification is that the analysis of some sanity checks pinpointed a problem with ARCH2 that is not reported in AIR6110. The problem is caused by the fact that the accumulator is positioned downstream of the selector valve, so that a fault in the accumulator can cause inadvertent braking. The problem is only reported for ARCH3; ARCH2BIS was included in the analysis to correct the problem.

Fault-tree analysis We now consider the construction of Fault Trees for each of the architectures and requirements, from the models obtained with the model extension

functionality of XSAP, as described in Section 3. Each TLE is the violation of a system requirement. In order to cope with scalability issues, we limited the space of the problem in two ways: restricting the set of faults, and limiting the cardinality of the cut sets. This follows a standard practice in traditional safety analysis: given the manual effort required, priority is given to cut sets of lower cardinality or greater likelyhood.

The analyses have been run on the five architectures, for all safety properties and two additional properties. For each property, cardinality goes from 1 to 5, and also with no restriction. In addition to the complete set of faults, six different restricted sets of faults have been defined and observed. In total, 3150 fault tree constructions have been launched. Overall the activity resulted in 3089 computed fault trees and 61 computations timed out. The fault trees have minimal cut sets ranging from 0 to tens of thousands. All fault tree constructions have been performed using IC3 engine on a cluster of 64-bit Linux machines with CPU going from 2.4 Ghz to 2.7 Ghz Intel Xeon, with a memory limit set to 30Gb and a time limit of 10 hours.

For lack of space, we report only a sample of the results obtained for ARCH4. A detailed account of the results for all architectures, as well as the formal models and artifacts produced by the analyses, are available at [2]. Table 3 reports the number of minimal cut sets for the 15 chosen properties, for the full set of faults, with cardinality going from 1 to 5. The last column indicates whether the computation has been completed without cardinality bound (Y) or if it timed out (N). We also report the probability (Prob.) for each of the top-level events, for an association of the basic faults with a probability (in the N cases, the reported value is a lower bound). The execution time required to generate results ranges from seconds (for fault tree with dozens of minimal cut sets) to minutes or hours (for fault tree with thousands of minimal cut sets).

The problem was also tackled by means of contract-based safety analysis [3]. Given the inherent scalability of the contract based approach, we were able to produce hierarchical fault trees (HFT). The fault trees have been produced in a few minutes for all top level contracts for the full configurations. As discussed in [3], the compositional approach produces hierarchical fault trees whose corresponding set of minimal cut sets is an over-approximation of the one obtained with the monolithic approach. This was confirmed in the experiments. We also notice that over-approximation is a common practice in safety analysis. The two approaches can be considered complementary.

Architectures Comparison We carried out a global comparison of the architectures, based on the results obtained for each of them. Basically, the findings confirmed the weaknesses of ARCH1: its number of "single points of failure", i.e. minimal cut sets of cardinality 1, is always greater, or equal, than in the other architectures. The probabilities associated to the TLEs concerning the loss of wheel braking (S18-WBS-R-0321, S18-WBS-R-0322-left/right) are also greater than in the other architectures. The probability associated to the inadvertent braking of one wheel without locking (S18-WBS-R-0325-wheelX) is better in ARCH1 than in the other architectures. This is due to the fact that even if there is the same number of minimal cut sets at lower cardinality, the components at fault are not the same and neither is reliability.

The fault trees for the pair ARCH2 and ARCH3 are the same, which suggests that the modification of the control system (i.e. the difference between the two architectures) has

Table 3. Fault Trees results for ARCH4 (- represents a timed out computation)

Arch/Prop		Prob.	$\|mcs\| = 1$	$\|mcs\| = 2$	$\|mcs\| = 3$	$\|mcs\| = 4$	$\|mcs\| = 5$	Full
Arch4	S18-WBS-R-0321	4.51e-10	0	6	627	629	-	N
	S18-WBS-R-0322-left	1.00e-05	2	2	203	46287	-	N
	S18-WBS-R-0322-right	1.00e-05	2	2	203	46287	-	N
	S18-WBS-0323	0	0	0	0	0	0	N
	S18-WBS-R-0324	2.50e-11	0	1	0	2	8729	N
	S18-WBS-R-0325-wheel1	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel2	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel3	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel4	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel5	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel6	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel7	1.20e-04	9	12	2596	0	0	Y
	S18-WBS-R-0325-wheel8	1.20e-04	9	12	2596	0	0	Y
	cmd implies braking w1	1.13e-04	13	30	7428	3815	1768	Y
	braking implies cmd w1	1.25e-04	10	24	2647	4530	59	Y

no impact on the safety; same observations hold for the pair ARCH2BIS and ARCH4. This is to be expected, since the change is triggered by a trade study aiming at reducing costs and easing maintenance, but the two control systems are designed according to the same redundancy principles, i.e. double Control Unit. The difference is that in one case the two CU's can be physically positioned in different places, while in the other they are part of a unique subsystem (which can, in very rare situations, break the assumption of independence of the two CU's). Common Cause Analysis, in particular Zonal Safety Analysis (ZSA), could confirm this point, and will be part of future activity.

The superiority of ARCH2BIS on ARCH2 (and similarly of ARCH4 on ARCH3) is demonstrated by a lower number of minimal cut sets with cardinality greater than 1. For the TLEs concerning the loss of wheel braking (S18-WBS-R-0321, S18-WBS-R-0322-left/right), the lower number of minimal cut sets appears at cardinality 3. For the TLE concerning the inadvertent braking of all wheels with locking (S18-WBS-0323), there is no difference up to cardinality 5. Concerning the inadvertent braking of all wheels (S18-WBS-R-0324), the lower number of minimal cut sets appears at cardinality 4. For the TLEs concerning the inadvertent braking of one wheel without locking (S18-WBS-R-0325-wheelX) the lower number of minimal cut sets appears at cardinality 2.

6 Conclusions, lessons learned and further work

We presented a complete formal analysis of the AIR6110 [25], a document describing the informal design of a Wheel Brake System. We covered all the phases of the process, and modeled the case study by means of a combination of formal methods including contract-based design, model checking and safety analysis. We were able to produce modular descriptions of five architectures, and to analyze their characteristics in terms of a set of safety requirements, automatically producing over 3000 fault trees, as well as quantitative reliability measures. We remark that one of the analyzed architectures (ARCH2BIS) was the result of detecting an unexpected dependency in the phases of the AIR6110. Specifically, the trade study on the control system (leading from ARCH2 to ARCH3) was carried out on an architecture suffering from a misplaced position of the

accumulator (fixed in ARCH4). In the following, we discuss some lessons learned, and outline directions for future activities.

Lessons learned The value in going from an informal description to a formal model was clearly recognized: the AIR6110 omits important information that is assumed to be background knowledge. The ability to produce the artifacts of the traditional design flow (e.g., architectural diagrams for visual inspection, fault trees) supported the interaction with subject matter experts, who were able to provide fundamental information to increase the accuracy of the models.

MBSA is a fundamental factor for this kind of application. First, it provides for automated construction of models encompassing faults from models containing only nominal behaviours. Second, traditional verification techniques, that allow to prove or disprove properties, are not sufficient: the automated synthesis of the set of minimal cut sets (i.e. the configurations causing property violations) is required to support the informal process and to provide a suitable granularity for the comparison of various architectural solutions. This approach also provides strong support for trade studies.

A key factor is the availability of automated and efficient engines. IC3 and its extensions to the computation of minimal cut sets allowed for the analysis of architectures that were completely out of reach for BDD-based safety assessment algorithms.

The use of an architectural modeling language, as proposed in the Contract Based approach supported by OCRA (and its integration with NUXMV), allows to reuse both models and contracts. For example, the similar architectures (e.g., ARCH3 and ARCH4) share a very large part of their models. This also makes it possible to analyze architectural variants with moderate effort.

There is a fundamental role for contract-based design. Its key advantages are the ability to mimic the informal process, thus ensuring traceability, and to support proof reuse. Contract-based design also supports the construction of Hierarchical Fault Trees, which are a fundamental artifact compared to the flat presentation of the set of minimal cut sets. The CBSA approach outlined in [3] enables for hierarchical fault tree generation, which are much easier to compute, and exhibit more structure when compared to a flat presentation of minimal cut sets. The open problem is how to evaluate the amount of approximation associated with the method.

Future work We will continue this work along the following directions, also driven by the findings in the case study. We will explore the use of alternative and more expressive modeling formalisms that may be more adequate to describe systems at a higher level of detail. For example, we will consider the use of SMT and more expressive logics, both on discrete and hybrid traces [8].

Contract-based design poses important challenges in terms of debugging. In particular, there is a need for suitable diagnostic information to support contract formulation (e.g., to understand why a certain contract refinement does not hold).

Another direction concerns increasing scalability for safety analysis. Realistic cases require the analysis of tens of thousands of minimal cut sets. We will investigate techniques to gain efficiency by introducing approximations (e.g., limiting cardinality and likelihood of cut sets); an important requirement will be the ability to calculate the degree of approximation of the result.

References

1. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP Safety Analysis Platform. ArXiv e-prints (2015)
2. Bozzano, M., Cimatti, A., Fernandes Pires, A., Jones, D., Kimberly, G., Petri, T., Robinson, R., Tonetta, S.: AIR6110 Wheel Brake System case study. https://es.fbk.eu/projects/air6110
3. Bozzano, M., Cimatti, A., Mattarei, C., Tonetta, S.: Formal safety assessment via contract-based design. In: Cassez, F., Raskin, J.F. (eds.) Automated Technology for Verification and Analysis, LNCS, vol. 8837, pp. 81–97. Springer (2014), http://dx.doi.org/10.1007/978-3-319-11936-6_7
4. Bozzano, M., Villafiorita, A.: Design and Safety Assessment of Critical Systems. CRC Press (Taylor and Francis), an Auerbach Book (2010)
5. Bradley, A.R.: Sat-based model checking without unrolling. In: 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI). pp. 70–87 (2011), http://dx.doi.org/10.1007/978-3-642-18275-4_7
6. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: 26th International Conference on Computer Aided Verification (CAV). pp. 334–342. Springer (2014)
7. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A Tool for Checking the Refinement of Temporal Contracts. In: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 702–705 (2013)
8. Cimatti, A., Roveri, M., Tonetta, S.: Requirements validation for hybrid systems. In: 21st International Conference on Computer Aided Verification (CAV). pp. 188–203. Springer (2009)
9. Cimatti, A., Tonetta, S.: A property-based proof system for contract-based design. In: 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 21–28 (2012)
10. Cimatti, A., Tonetta, S.: Contracts-refinement proof system for component-based embedded systems. Science of Computer Programming 97, 333–348 (2014)
11. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: Design, Automation and Test in Europe (DATE). pp. 1023–1028 (2011)
12. ESACS: The ESACS Project (Last retrieved on May 20, 2015), www.transport-research.info/web/projects/project_details.cfm?ID=2658
13. (FAA), F.A.A.: Advisory Circular (AC) 20-174. http://www.faa.gov/documentLibrary/media/Advisory_Circular/AC%2020-174.pdf
14. (FAA), F.A.A.: Advisory Circular (AC) 23-1309-1E. http://www.faa.gov/documentLibrary/media/Advisory_Circular/AC%2023.1309-1E.pdf
15. (FAA), F.A.A.: Advisory Circular (AC) 25.1309-1A. http://rgl.faa.gov/Regulatory_and_Guidance_Library/rgAdvisoryCircular.nsf/list/AC%2025.1309-1A/$FILE/AC25.1309-1A.pdf (1988)
16. FBK: nuXmv: a new eXtended model verifier, available at http://nuxmv.fbk.eu
17. FBK: OCRA: A tool for Contract-Based Analysis, available at http://ocra.fbk.eu
18. FBK: xSAP: eXtended Safety Analysis Platform, available at http://xsap.fbk.eu
19. ISAAC: The ISAAC Project (Last retrieved on May 20, 2015), http://ec.europa.eu/research/transport/projects/items/isaac_en.htm
20. Joshi, A., Heimdahl, M.: Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In: Winther, R., Gran, B., Dahll, G. (eds.) Proc. Conference on Computer

Safety, Reliability and Security (SAFECOMP). LNCS, vol. 3688, pp. 122–135. Springer (2005)
21. Joshi, A., Whalen, M., Heimdahl, M.: Model-Based Safety Analysis Final Report. Tech. Rep. NASA/CR-2006-213953, NASA (2006)
22. MISSA: The MISSA Project (Last retrieved on May 20, 2015), www.missa-fp7.eu
23. SAE: ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (1996)
24. SAE: ARP4754A Guidelines for Development of Civil Aircraft and Systems (2010)
25. SAE: AIR 6110, Contiguous Aircraft/System Development Process Example (2011)
26. Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick III, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. Tech. rep., NASA (2002)