Model Checking at Scale: Automated Air Traffic Control Design Space Exploration *

Marco Gario[1], Alessandro Cimatti[1], Cristian Mattarei[1], Stefano Tonetta[1], and Kristin Yvonne Rozier[2]

[1] Fondazione Bruno Kessler, Trento, Italy,
{gario,cimatti,mattarei,tonettas}@fbk.eu
[2] Iowa State University, Iowa, USA,
kyrozier@iastate.edu

Abstract. Many possible solutions, differing in the assumptions and implementations of the components in use, are usually in competition during early design stages. Deciding which solution to adopt requires considering several trade-offs. Model checking represents a possible way of comparing such designs, however, when the number of designs is large, building and validating so many models may be intractable.
During our collaboration with NASA, we faced the challenge of considering a design space with more than 20,000 designs for the NextGen air traffic control system. To deal with this problem, we introduce a compositional, modular, parameterized approach combining model checking with contract-based design to automatically generate large numbers of models from a possible set of components and their implementations. Our approach is fully automated, enabling the generation and validation of all target designs. The 1,620 designs that were most relevant to NASA were analyzed exhaustively. To deal with the massive amount of data generated, we apply novel data-analysis techniques that enable a rich comparison of the designs, including safety aspects. Our results were validated by NASA system designers, and helped to identify novel as well as known problematic configurations.

1 Introduction

When multiple system design configurations are possible, there is a need to map the design space in order to understand the big picture, and be able to demonstrate the impact of design choices, such as different combinations of potential subcomponents with different features, on the overall functionality and safety of the system. Safety assessment of complex and critical systems can clearly benefit from the use of formal methods techniques [28,34,24,20,27,29,15,30,14,22,31], but a large space of possible designs presents major challenges for model-checking

* Thanks to the Flight Trajectory Dynamics and Controls Branch of NASA Ames Research Center and NASA's Functional Allocation Project for supporting this work. All models and specifications are available at https://es-static.fbk.eu/projects/nasa-aac/.

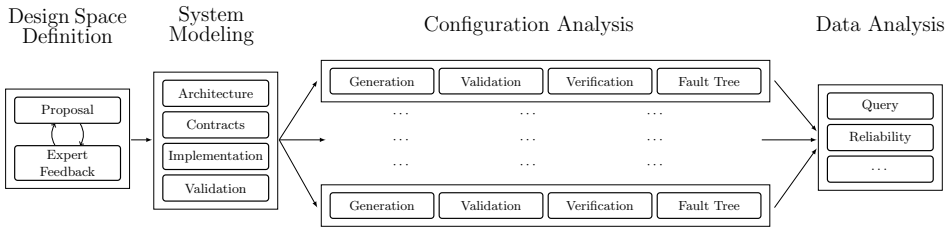

Fig. 1. Process Overview

analysis, including producing models of each design, cross-design validation, and comparative safety analysis across the large design space. We address these challenges, exemplifying our methodology on NASA's full-scale design space for NextGen air traffic control, in which there are many ways to allocate essential functions such as aircraft separation assurance [26], and competing possible implementations of the same components. The U.S. government made NASA primarily responsible for the design and verification of NextGen air traffic control [3,2]. The new air traffic management system is expected to be in place for decades to come [5] so we must evaluate the design space thoroughly to ensure that we guarantee safety while allowing optimization for important secondary considerations. The importance of early-design stage optimization carries to many classes of critical or long-lived projects, including commercial aircraft and space missions, where the need to change the design later in the system development process would be extremely difficult, and very costly.

In this paper, we discuss the application of model-checking-based techniques to support the exploration of the NextGen design space. This is one of six studies funded within NASA's Functional Allocation Project and will contribute directly to the final system design. We define a compositional, parameterized modeling framework that can generate more than 20,000 possible designs. In collaboration with NASA Ames and NASA Langley experts, we focus in on the 1,620 that they identified as the most instructive configurations for a comparative analysis. The outcome of this analysis provides significant insights into the features of the various configurations. In order to tackle the huge design space, we develop a new process that relies on multiple tools. The activities, depicted in Figure 1, can be summarized in four main phases: *Design Space Definition*, *System Modeling*, *Configuration Analysis*, and *Data Analysis*.

Design Space Definition. The stage was set by working with NASA in order to identify precisely (yet informally [26]) the situations of interest, and by defining the modeling dimensions to capture them.

System Modeling. Modeling each solution independently would be too time-consuming (if not outright unfeasible). Plus each model needs to be properly validated to ensure that it upholds the expected properties. Furthermore, independent models would require a lot of maintenance effort to propagate changes and ensure that they are all aligned with NASA's most current designs. We can manage these sources of complexity by combining several ingredients. First, we

use an architectural language (i.e., OCRA [17]) to separate the system architecture from the implementation of the single components (obtained as SMV [16] models). This allows us to model each component in isolation, partitioning the effort, and minimizing the time required to validate changes in any component. Additionally, this permits changes to the implementation of a single component without impacting the rest of the system. Second, we use contracts (encoded in OCRA as LTL formulas) to characterize each component. This allows us to properly specify the interactions between components, and decompose the validation properties into more localized subcomponent properties. Third, we use parameters to factor out multiple configurations into a single (although more complex) model. If two configurations require only marginal changes to an implementation, we capture these changes using parameters within the models. These techniques allow us to automatically generate a formal representation for each configuration in the design space, with great confidence in their correctness and alignment.

Configuration Analysis. We verify each model against the properties of interest; in addition, techniques for safety assessment identify which combinations of faults lead to the violation of fundamental properties. The corresponding Fault Trees are automatically computed (using xSAP [10]), thus providing additional information on the reliability of each configuration. We instantiate and analyze each configuration independently, exploiting the typical parallelism of modern computing infrastructures, thus significantly speeding up the analysis.

Data Analysis. Such analysis results in a significant amount of data, and poses the problem of how to analyze it. We combine this data into a symbolically represented dataset, linking each configuration to its satisfied properties and Fault Trees. This dataset is particularly useful in such an exploratory phase, since it describes the whole design space and can be studied offline. For example, by automatically extracting sets of configurations enjoying specific properties (e.g., absence of single points of failure), it is possible to achieve a better understanding of the design space. Our focused analysis of NASA's air traffic control design space confirmed expected results [25,23] as well as identifying novel ones. In particular, we highlighted the need for additional assumptions when dealing with changes in delegation of separation assurance from an aircraft to the ground, e.g., in case of a request for backup.

The contribution of this paper is twofold. First, we develop a complex and realistic case study of public relevance, and make models, tools, and results publicly available for future investigation (at [4]). This is no ordinary case study, and to be able to handle the massive size, we need to exploit a novel process that is our second contribution. Our process is able to scale to address a large design space exploration problem. The process builds on existing tools and techniques and adds a novel data analysis phase that is necessary to obtain insights from the large amount of generated artifacts. We show that this technology is mature and able to assist designers in formalizing and narrowing down design choices in an early phase of system design.

The rest of the paper is structured according to the process described above (Figure 1). Sections 2 to 5 illustrate each phase of the process in greater detail. Related works are discussed in Section 6, and Section 7 concludes with possible directions for future work.

2 Design Space Definition

The main objective of an air traffic control system is to avoid aircraft collisions. In air traffic management, a Loss of Separation (LOS) between two aircraft occurs when they are predicted to pass too close to each other. One of the major goals of the next generation of air traffic control is to minimize the number of times that a LOS ever occurs. This task is called *Separation Assurance*. In this case study, we are interested in studying the separation assurance provided by different designs when splitting the functionality between components on-board airplanes and on-ground. In particular, aircraft that always rely on the ground for separation assurance are called Ground Separated (GSEP), while aircraft with on-board separation assurance capabilities are called Self-Separating (SSEP). The main distinction between the two types of aircraft is the ability of SSEP to perform self-separation, without the need of contacting the ground. The goal of distributing the responsibility for separation assurance across different components is to increase efficiency and improve fault tolerance.

Our work started by considering several proposals from NASA's *Flight Dynamics, Trajectory, and Controls* Branch for different solutions regarding Function Allocation for Separation Assurance [26]. These ideas were the result of considering several features and characteristics in a preliminary phase.

Our first step was to identify and formalize the dimensions shared by different proposals, and this allowed us to define the design space. In order to model the airspace and its dynamics, we track each aircraft's intended trajectory through four different time-windows: *Current*, *Near*, *Mid*, and *Far*. These indicate increasingly distant points in time. For each window, we encode the intended position of the aircraft. However, since we are only interested in whether two aircraft can potentially be in a conflict, we simplify this information. For a given time-window, we say that two aircraft are in the same *Conflict Area* (CA) *iff* their trajectories are too close to each other and would cause a Loss of Separation. We say that two aircraft are in LOS *iff* they are in the same conflict area in the Current time-window. If two aircraft are in the same CA in another window, we say that they have a *predicted* LOS. These abstractions make it possible to focus on the other modeling dimensions: what information the different agents share, how they behave in case of predicted LOS, and the impacts of the actions of each agent on the overall system. Contrary to previous works (e.g., [28]), we consider more complex interactions between separation agents, components with multiple implementations, and priorities in case of predicted LOS. We derived six modeling dimensions that enable us to capture these different trade-offs:

1. SSEP Separation Agent
2. Aircraft Mix

3. Information Sharing
4. Burdening Rules
5. Communication Steps
6. ACDR Implementations

SSEP Separation Agent. A key difference between the solutions is who is responsible for performing separation for the SSEPs. We split this task into separation for the Tactical (Near- and Mid-) and Strategic (Far-) windows. For each of these windows we define who is in charge of separating the SSEPs: the ground (ATC), the aircraft ($SELF$), or the aircraft with possible delegation to ground ($SATC$). If the ground ATC is in charge of separating the SSEPs, then it computes the resolutions and sends them to the aircraft. If the aircraft is in charge of its own separation, computation of a resolution strategy happens on-board, possibly involving coordination between aircraft. The third case ($SATC$) captures the possibility for an SSEP to delegate its own separation to the ground. This is used to capture different situations such as backup in case of a fault, privileged traffic corridors, and transfer of responsibility in designated airspace regions. In the future, we expect other cases to be studied. For example, resolutions might be computed on-board but require approval from ground.
Aircraft Mix. We consider situations in which all aircraft are of the same type, and also where mixed types coexist. The same design can be analyzed without SSEPs, with an even number of GSEPs and SSEPs, without GSEPs, or any option in-between. Each combination is indicated by the number of GSEPs and SSEPs, i.e., $\langle \#GSEP, \#SSEP \rangle$.
Burdening Rules A priority must be defined in order to address detected conflicts between aircraft of different types. Burdening rules define who should move when such a conflict occurs: 1) *Undefined*, 2) *GSEP*, 3) *SSEP*. For example, if the burden is on the GSEP, then the conflict should be resolved by changing the trajectory of the GSEP. If the burdening rules are undefined, then each agent will arbitrarily choose a burdened strategy, and consistently apply it to every conflict.
Information Sharing. It is important to consider the minimization of required communications, in order to reduce reaction times and system complexity. Therefore, we need to understand what is the minimum amount of intent that aircraft need to share. We make two main distinctions: information sharing from *GSEPs to SSEPs* and from *SSEPs to ATC*. For each of these two information sharing pipelines, we consider scenarios from sharing no information (*None*) to sharing information concerning just the *Current*-window, up to the *Near*-window, up to the *Mid*-window, or all the windows (*Far*-window).
Communication Steps. In some situations, multiple communication rounds might be needed in order to reach an agreement among the parties. However, delays in communication and availability of the networks make it necessary to minimize the number of communication rounds that need to occur.
ACDR Implementations We considered different implementations for the Airborne Conflict Detection and Resolution (ACDR) component. The simplest implementation of the ACDR computes a resolution without considering the be-

Table 1. Summary of possible and considered design dimensions

Name	Possible		Considered	
	Values	Size	Values	Size
SSEP TS SA	ATC, SELF, SATC	3	ATC, SELF, SATC	3
SSEP SS SA	ATC, SELF, SATC	3	ATC, SELF, SATC	3
Aircraft Mix	$\langle 4,0\rangle$, $\langle 3,1\rangle$, $\langle 2,2\rangle$, $\langle 1,3\rangle$, $\langle 0,4\rangle$	5	$\langle 4,0\rangle$, $\langle 3,1\rangle$, $\langle 2,2\rangle$, $\langle 1,3\rangle$, $\langle 0,4\rangle$	5
Burdening Rules	Undef, GSEP, SSEP	3	Undef, GSEP, SSEP	3
GSEPs to SSEPs Info	None, Current, Near, Mid, Far	5	Current, Far	**2**
SSEPs to ATC Info	None, Current, Near, Mid, Far	5	Far	**1**
Com Steps	1, 2, ...	2	1, 2	2
ACDR Implementations	Simple, Asymmetric, Non-Receptive	3	Simple, Asymmetric, Non-Receptive	3
TOTAL		20,250		1,620

havior of the other aircraft ("ACDR Simple"). A more complex implementation, instead, takes into account how the other SSEPs are going to resolve the conflict, and uses this knowledge to compute a resolution that is guaranteed to solve the current conflict ("ACDR Asymmetric"). Finally, the last implementation (called "ACDR Non-Receptive") is the one in which we declaratively enforce the assumption that conflicts among SSEPs will be resolved without specifying how, thus constraining the environment with a non-receptive specification [6]; this last option is useful to study the system behavior assuming a perfect ACDR.

Table 1 shows the possible dimensions defined during the first analysis, and yields a design space with 20,250 configurations. Though we can scale to automatically generate and analyze this many models, further discussions with NASA domain experts led us to focus our exhaustive analysis on the subset of 1,620 configurations most interesting from the domain point of view. In particular, they decided to fix the information sharing of the SSEPs in order to provide all information (i.e., *Far*) and consider only the two extreme cases for the information shared by the GSEPs: *Current* and *Far*. This reduced the design space to a set of 1,620 configurations (right part of Table 1). These are the configurations analyzed in the rest of the paper.

3 System Modeling

The dimensions described in Table 1 are captured by defining a unified structure including all possible configurations. This structure is equipped with parameters and multiple implementations of the components, making it possible to model the whole system once, and then automatically generate any of the 1,620 possible instances. This reduces the modeling effort that is, in terms of resources, the most expensive part of the process. However, we need to pay particular attention to the validation of the instantiated models, in order to make sure that all expected behaviors are properly captured.

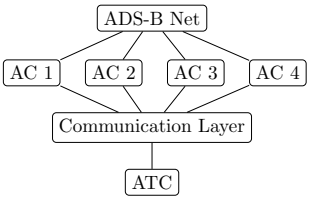

Fig. 2. Model Architecture

The general structure of the model is shown in Figure 2, and includes four aircraft, the ATC, and two different types of networks: ADS-B and Communication Layer. ADS-B is used only among the aircraft, while the Communication

Layer is used between the aircraft and the ATC. This choice makes it simple to provide different characteristics to the two networks: faults, symmetry, amount of information, delays, etc. We always consider up to four aircraft instances. This is sufficient to capture all combinations of conflicts between aircraft of different types: GSEP-to-SSEP, GSEP-to-GSEP, SSEP-to-SSEP. This abstraction only represents how many aircraft can be in a single conflict at the same time, and does not assume anything about the size of the airspace [34].

Figure 3 shows the decomposition of the system into a hierarchy of component types, and this provides an architecture that can be incrementally refined. For example, we break down the definition of the Aircraft and ATC components into subcomponents, and this compositional approach allows us to simplify modeling and validation.

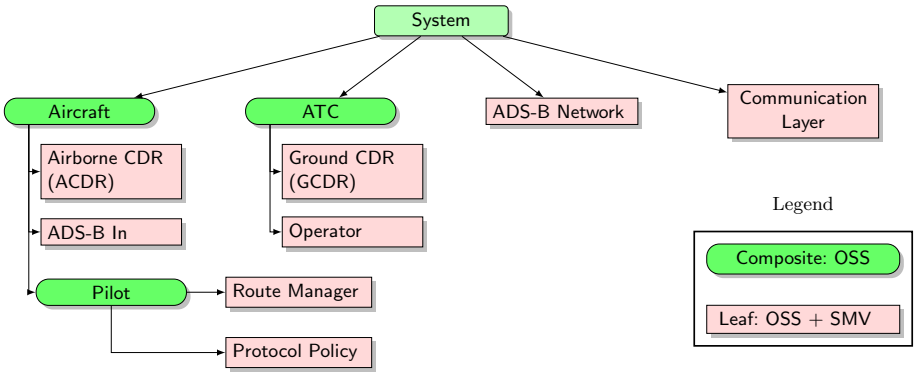

Fig. 3. Hierarchical decomposition

We use the Aircraft component (the most complex component) to exemplify our parametric modeling approach. There are two types of aircraft: SSEP and GSEP. Since these two types differ only in few ways, they are modeled as a generic aircraft component whose behavior is selected via a set of parameters, as listed in Table 2. More specifically, we model the *Aircraft* component as having the following parameters: `adsb_in`, `ts_agent`, `ss_agent`, and `burdening`. The parameters `ts_agent` and `ss_agent` are used to specify who is in charge of the Tactical Separation (TS) (i.e., Near- and Mid-window) and Strategic Separation (SS) (i.e., Far-window). Similarly, the parameters `burdening` and `adsb_in` capture, respectively, the information about the burdening rule in use and the availability of the ADS-B receiver. Using this parametric model, we can describe a GSEP as an aircraft that is always separated by ground, and that does not have an ADS-B In component:

```
Aircraft(adsb_in=No, ts_agent=ATC, ss_agent=ATC, burdening=GSEP).
```

The impact of parameter choice is localized to the parameterized subcomponent. For example, the `burdening` parameter has an impact only on the ACDR compo-

Table 2. Parameters, Inputs and Outputs of the Aircraft model

Type	Name	Domain
Parameter	id	[1..4]
	adsb_in	Boolean
	ts_sa_agent	{ATC, SELF, SATC}
	ss_sa_agent	{ATC, SELF, SATC}
	burdening	{Undefined, GSEP, SSEP}
Input	suggestion_{near,mid,far}_ground	Conflict Area [0..4]
	communication_phase	Boolean
	ac_{1,2,3,4}_intention_{current,near,mid,far}	Conflict Area [0..4]
	ac_{1,2,3,4}_{ts,sa}_agent	{ATC, SELF, SATC}
Output	intention_{current,near,mid,far}	Conflict Area [0..4]
	predicted_conflict_{near,mid,far}	Boolean
	request_{ts,ss}_sa_ground	Boolean

nent. Having components whose implementations are independent of the model's parameters makes it possible to *re-use* these components for multiple configurations. We also use a similar approach for modeling faults in the communication networks, and we localize all of those faults within the network components: ADS-B Network and Communication Layer. As shown in Figure 3, there are two different components that are used to capture the ADS-B functionality: the *ADS-B Network* and the *ADS-B In* component. By separating these (conceptually related) components, we are able to model the aircraft independently of the faults, and number of aircraft connected to the network. Table 2 provides a summary of the input and output information, and of the parameters for the Aircraft component. In each configuration, we enforce that all GSEPs must have the same parameters, and this applies also for the SSEPs. Therefore, in the same configuration there cannot be two SSEPs with, e.g., two different separation assurance agents. This is not a limitation of the model or tools, but a design choice motivated by the domain that we are exploring and our choice to keep the model more understandable and limit the scope to realistic scenarios.

The architecture shown in Figures 2 and 3 is captured using the OCRA language [17]. Breaking components (e.g., Aircraft) into simpler components simplifies both modeling and validation. In particular, we can write properties about the Aircraft and then decompose them into properties of the subcomponents. This pattern is called *Contract-Based Design*, and it is supported by OCRA using contracts expressed in Linear Temporal Logic (LTL). For example, we write a contract for the Aircraft (Figure 4) and decompose it into contracts on its subcomponents (see `REFINEDBY` in the Figure). To take advantage of contract-based design we need to perform two steps [18]. First, we need to check that the refinement of the contract is correct. This means that the guarantees provided by the subcomponents in the refinement are sufficient to prove the guarantee of the supercomponent. After performing this step, we know that independently of the choice of parameters, if the implementations of the ACDR and Pilot satisfy

```
CONTRACT AC_maintain_intention_ts_self
-- If self-separating, during communication phase if no conflict
-- is predicted, the intention will not change.
-- Tactical Separation Case.
assume: TRUE;
guarantee: always ((communication_phase and ts_sa_agent = SA_SELF) implies
    ((not predicted_conflict_near implies
          next(intention_near) = intention_near) and
     (not predicted_conflict_mid implies
          next(intention_mid) = intention_mid)));

CONTRACT AC_maintain_intention_ts_self
REFINEDBY cdr.ACDR_no_conflict_means_maintain_near,
          cdr.ACDR_no_conflict_means_maintain_mid,
          pilot.Pilot_apply_ts_self,
          pilot.Pilot_intention_is_not_nop;
```

Fig. 4. Example of a contract on the Aircraft component

their contracts, then also the Aircraft satisfies its contract. As a second step, we verify that the implementations of each component satisfy their contracts. This operation is done locally on the component in isolation and, since most components are relatively small, it can be performed efficiently. Every time we modify a basic component, we only need to validate it against its contracts, and we are guaranteed that the composite components will still satisfy their contracts. This way of using contracts significantly speeds up the design loop. To draw a parallel with software engineering, the contracts that we write are comparable to unit tests in which we focus on the correctness of the component in isolation.

An added benefit of this process of contract decomposition is that it requires a rigorous understanding of the relationships between the components. This raises interesting questions about how to define the components, how to divide responsibilities, and what behavior can be expected by every component in nominal situations. In fact, we are forced to define requirements that all component implementations must satisfy. In our case, this investigation was supported by a close collaboration with NASA, which resulted for example, in the definition of multiple possible ACDR implementations, and the definition of more than 130 contracts.

4 Configuration Analysis

Once the unified model is complete, we proceed to analyze each possible configuration in isolation. For each configuration we break the analysis into the following steps:

1. Instance Generation
2. Airspace, Nominal, and Extended Validation
3. Nominal and Extended Verification
4. Fault Tree and Reliability Analysis

Automation of this phase is very important. Each step is run automatically, from the definition of the instance to the generation of all verification and Fault Tree artifacts. This ensures that the process is reproducible and scalable.

Instance Generation Each leaf component in our hierarchical architecture is associated with an implementation (a behavioral model defined as an SMV file) by defining a *map* file. The OCRA tool uses this mapping to generate a single monolithic SMV file of the instance. This makes it extremely easy to instantiate the system with multiple functional implementations of the components, and also to create instances with and without faults. We pass parameters through the OCRA architecture using pre-processing instructions to define constants. In this way, the variability of the model is limited to the OCRA architecture and map files used during the generation phase. The outcomes of this phase are three models: *airspace*, *nominal*, and *extended*. These are standard SMV files, without parameters, that can be analyzed by any out-of-the-box technique.

Airspace, Nominal, and Extended Validation The models for the configuration are generated automatically, therefore, before proceeding to the verification step, we need to gain confidence in the quality of the generated models. For this reason, we perform these additional validation steps.

The *airspace* model captures the system without separation assurance agents. This is the first validation check: the model must allow the occurrence and resolution of LOS. We generated this model by mapping the separation agents to implementations that have no constraints, while using nominal implementations for the aircraft and networks. To certify that the components work correctly together, we verify 18 CTL properties encoding the possibilities of bad and good behaviors, and 24 LTL properties derived from contracts.

The *nominal* model uses a nominal implementation for every component, including separation agents. Unlike the extended model, in this case we do not allow components to fail. We validate this model with 29 LTL properties derived from the contracts of the components.

Finally, the *extended* model uses an implementation for every component that includes faults (95 faults in total, as described in [28]). The validation of the extended model checks that all faults are possible (through 137 CTL possibility properties), and that they respect their dynamics, i.e., permanent or transient, with 29 LTL properties.

Overall, the validation of the 3 models requires a combination of different techniques in order to be effective and be carried out in a limited time. The CTL verification requires a fixpoint-based approach, using BDDs, while for the LTL properties, we use the IC3-based algorithms implemented in nuXmv [16]. Every property is checked against a known result that, if violated, causes the analysis to stop for further investigation.

Nominal and Extended Verification In this step, we characterize different configurations by verifying additional properties. The most important is whether LOS can always be avoided (NO-LOS), followed by stronger versions: NO-LOS-Near, -Mid, -Far. Other properties provide additional information on the quality of the configuration, e.g., *Detect-Near* "Every conflict in the Near-window (Mid-, Far- respectively) is detected by at least one Agent." This property is satisfied if the ATC (which is an Agent, in this context) detects a conflict between two

aircraft, without either of the aircraft detecting it. It is clear that we can devise stronger versions of this property, and apply it to different time-windows (e.g, Detect-Mid, -Far). This provides a simple way of ranking configurations according to how many and which properties they satisfy. During extended verification, we check instead whether these properties are still satisfied in the presence of faults. For most properties this will not be the case. However, if some property is satisfied even with faults, it means that the property and the faults have no relation in the given configuration. In this step, we verify 24 LTL and 30 invariant properties on both the nominal and extended models.

Fault Tree and Reliability Analysis We compute the Fault Tree associated with each safety property in order to understand the resilience of each configuration in the presence of faults. Fault Trees are a standard in safety-critical domains [32,7,8]. More specifically, we compute the set of minimal cutsets, i.e., all possible faults configurations (called cutsets) that can cause the violation of the given property. These cutsets are minimal because they only include the faults that are necessary to violate the property. Minimal cutsets are computed automatically from the formal model, using the IC3-based technique described in [11] and implemented in xSAP [10]. For each Fault Tree, we also generate a *reliability* function [12]. This function relates the probability of violating the property to the probability of occurrence of each basic fault.

5 Data Analysis

Each configuration can be analyzed independently. We exploit this fact and run the analysis on a cluster with 12 Intel Xeon X5650 processors (72 cores). The average size of the models was 10^{107} states, and each model was checked against 346 properties. The two most difficult steps were those of model validation, due to the need for BDD-based reasoning, and minimal cutset computation, since it requires solving a parameter synthesis problem. These two steps were completed within an hour for most configurations, but for roughly 10% of the models, they required several hours to complete. Verification of the LTL properties was performed using the nuXmv [16] IC3 implementation, requiring roughly 5 minutes per model.

Once all results are available, we can perform the last step of the process: *Data Analysis*. Each configuration provides us with a set of verification results and a set of Fault Trees. Therefore, we face the challenge of how to intuitively represent the information provided by more than 1,600 Fault Trees and verification results. We approached the problem by collecting these artifacts into relations. The first, $V \subseteq C \times \mathbb{B}^n$, relates each configuration (i.e., a set of values for the parameters) to the satisfaction of the verification properties. The second, $FT \subseteq C \times \mathbb{N} \times 2^{MCS}$ instead relates each configuration and property index to the set of minimal cutsets (MCS) associated with it. This data can be queried and manipulated offline, by the domain experts, in order to obtain more insights into the design space.

5.1 Summary of Results

Most of the configurations (Table 3) satisfy the key property of avoiding Loss of Separation (NO-LOS). The fact that NO-LOS-Far is satisfied by some SSEP-Only configurations is due to the non-receptive implementation of the ACDR, which assumes that trajectories are computed in a way that avoids potential conflicts in the Far-window. However, not all configurations using the SSEP-Only ACDR are immune to LOS. For example, when including burdening rules, GSEPs (that do not use the ACDR) can interfere with the SSEPs and lead to a LOS.

Table 3. Models satisfying NO-LOS for different windows

	GSEP-Only 4-0	Mixed 3-1	Mixed 2-2	Mixed 1-3	SSEP-Only 0-4	Total
NO-LOS	324	244	212	213	258	1251
NO-LOS-Near	324	244	209	210	252	1239
NO-LOS-Mid	324	192	138	141	198	993
NO-LOS-Far	0	0	0	18	84	102

Prime Implicants To extract interesting facts from the verification results, we synthesize the region of parameters that satisfy a property of interest. To compute the region of parameters that satisfy a property, we fix the property value and quantify away the other properties in the relation V. E.g., for NO-LOS:

$$NO\_LOS(C) = \exists P_1, \cdots, P_n.\ V(C, P_1, \cdots, P_n) \wedge P_{NO\_LOS}$$

where P_i is a Boolean variable associated with the verification result for property i, and C is the set of configuration variables (i.e., parameters). In this way, we can compute the region of parameters associated with the satisfaction of each property. Very few of these regions have a compact representation. To extract interesting facts from these regions, we compute the *prime implicants* of the region, i.e., the set of minimal elements that are sufficient to enforce the satisfaction of the property. For cardinality 1, we obtain the following implicant for NO-LOS:

$$(MIX = \langle 4, 0 \rangle) \vee (SSEP\_TS\_SA = ATC) \vee (SSEP\_SS\_SA = ATC)$$

This tells us that there are three ways to guarantee NO-LOS: (i) having only GSEP airplanes, or having the ATC in control of the (ii) Strategic or (iii) Tactical separation of any SSEP.

By checking that NO-LOS-Far is achieved only by configurations using non-receptive ACDR, we verified the corresponding claim from Table 3. Moreover, we verified that not all configurations using non-receptive ACDR can satisfy NO-LOS-Far, thus discovering a necessary but not sufficient condition. These analyses were performed using pySMT [21] in order to represent the data using BDDs [13] for efficient querying.

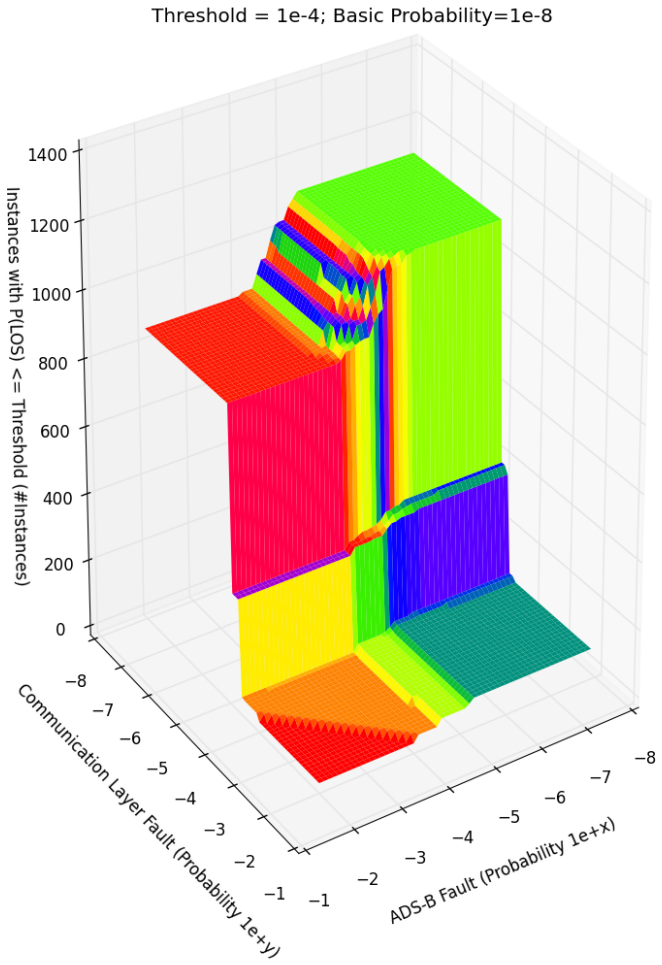

Fig. 5. Impact of the communication faults on LOS probability.

Reliability Functions Analyzing the reliability functions obtained from the fault trees, we can synthesize the region of configurations that have a probability of violating a property below a given threshold. This result provides us different sets of candidates that are able to guarantee a high reliability. In addition to that, we want also to analyze the impact of a variation in the probability of failure of different groups of components. In Figure 5 we demonstrate this last analysis by proceeding as follows. First, we partition the faults into three groups: the ones related to the *Communication Layer*, the *ADS-B*, and all the others. For each configuration and each value of the probability of the faults of the Communication Network (y axis) and of the ADS-B Network (x axis), we compute the probability of reaching a LOS, by considering all other faults to have a fixed probability of 10^{-8} (Basic Probability). In Figure 5, we summarize this information by plotting how many configurations have a probability of leading to a LOS that is below the threshold of 10^{-4} for the given probability of the faults. Interestingly, we see that reducing the reliability of the Communication

Layer has a bigger impact than reducing the reliability of the ADS-B network. We see this because when the probability of faults in the ADS-B is high (x axis close to -2), but the probability of fault of the communication layer is low (y axis close to -8), the probability of reaching a LOS is below the threshold of 10^{-4} for more than 800 configurations. If we look at the opposite situation, instead, we see that less than 100 configurations have a probability of reaching LOS that is below the threshold. The insight that we gain from this is that many of the analyzed configurations are robust with respect to failures of the ADS-B.

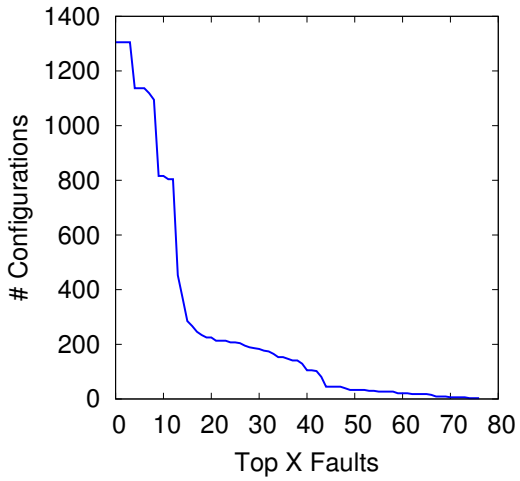

Fig. 6. Configurations impacted by the top N single points of failure.

A different analysis is presented in Figure 6 in which we analyze how many configurations share the same top N single points of failure. A *single point of failure* is a single fault that is sufficient to (in our case) cause a LOS, and corresponds to a minimal cutset of cardinality one. There are roughly 10 single points of failure that are shared by more than a thousand configurations. However, we also notice that most faults are single points of failure for a limited number of configurations; recall that there are 95 faults in total. If the probability of those 10 faults is very high, then we can significantly prune the design space, and focus only on the configurations that are not affected by those faults.

5.2 Interesting Executions

A selection of the most relevant results was discussed with the domain experts. In particular, we were able to independently reproduce two known issues, *side-walk* [33,25] and *coincidental* conflicts [23], and discover a new one.

Side-walk Conflict. Side-walk conflicts occur whenever we use the "simple" implementation of the ACDR, in which conflicts between SSEPs are resolved

by choosing a free conflict area. The problem occurs when more than one SSEP decides to move to the same conflict area. Due to the symmetry of the resolution algorithm, this strategy is not guaranteed to resolve the conflict. To break this symmetry, we developed the asymmetric version of the ACDR.

Coincidental Conflict. The asymmetric ACDR is not able to resolve conflicts early. In particular, we would like to always satisfy NO-LOS-Mid, i.e., avoid predicted LOS in the Mid-window. This is not possible if we allow only one communication step. In fact, if four aircraft are in two different conflicts that are resolved correctly, they might still end up in a new conflict. Consider the two conflict sets: {AC1, AC2} and {AC3, AC4}. AC1 and AC3 decide to move to solve their respective conflicts. However, they choose to move to the same conflict area. An additional round of communication is needed in order to resolve this conflict, and this generalizes to needing at most $log(n)$ communication steps when considering n aircraft.

Backup From Ground. The novel problematic configuration that we identified stems from limited requirements on the behavior of the backup operation, i.e., when an SSEP is able to request backup from ground and it delegates its separation to the ATC (SATC). This turned out to require more assumptions than were initially considered. In fact, when enabling this behavior, all configurations violate NO-LOS, excluding the ones with non-receptive ACDR. This is motivated (as shown by the counterexamples) by a lack of information and a mismatch of expectations in the airspace. In particular, in the design used in this project, whenever an aircraft requests ATC assistance, the other aircraft are not aware of it. Therefore, all of the other SSEPs expect the aircraft to maintain its behavior as an SSEP. In order to solve this issue, we propose two options. First, requests for ground-assistance are relayed to other aircraft. Second, the algorithm for separation used by ATC needs to take into account that the aircraft was an SSEP, and therefore compute a resolution taking into account what the other SSEPs expect the aircraft to do. These extensions are left as future work.

6 Related Work

Before NASA turned to the question of what designs were best for automated air traffic control, it was necessary to explore what designs were *possible*. To that end, NASA launched several initiatives to formally reason about a *single* such system; two of these works, using symbolic model checking [34] and probabilistic model checking [35] techniques led to the decision to use the former for the problem of broader design space exploration. However, neither technique proved sufficiently scalable to capture all of the relevant details of a single design at the same time.

This paper presents a large advancement along the same line of research of [28], in which a modeling abstraction for the problem was proposed, by designing and verifying a monolithic model. This modeling abstraction proved to be suitable for capturing the problem, however, it could not be scaled to cover the entire design space. Therefore, in this work we devise a tailored process that

allows us to model, validate, verify, and compare the full design space (i.e. 20,000 designs and beyond) with exhaustive analysis of the more than 1,600 most likely candidate designs. This was made possible by breaking down the modeling and using a compositional approach based on contract-based design (as opposed to the monolithic approach of [28]). The size of the design space not only created challenges for running the analysis, but also for analyzing the results: we had to consider new ways of looking at the considerable amount of data produced in order to extract interesting information rather than providing NASA with a firehose of data. Thanks to the extensive coverage of the design space enabled by the process described in this paper, we managed to identify some configurations that were of interest for NASA. The examples that we highlight show that this approach pinpointed implicit assumptions and critical points in the design process.

The term *design space exploration* is commonly used to describe the study of a design space (mostly combinatorial) by avoiding the computation of all solutions and optimizing with respect to some cost function. For example, Airbus [9] uses automated techniques to evaluate design spaces, in which multiple solutions are compared and sorted with respect to their weight. It is important to notice, however, that we are dealing with a sequential problem while works such as [9] deal with combinational ones. Moreover, the existence of a cost function allows the optimization engine to prune "bad" configurations, thus reducing the actual number of configurations that will be eventually checked. In our case, there is no cost function defined; we are instead interested in a better understanding of the design space, and thus want to be able to thoroughly analyze every possible design. Therefore, we analyze all of the realistic configurations and collect the data in a form suitable for subsequent comparison.

When we move from combinational to sequential problems, we find works related to product lines, e.g., Software Product Lines [19], that deal with a similar problem of verification of a parametric system. In [19] the authors propose an extension to NuSMV that is able to perform symbolic model checking of an extended version of CTL (feature-oriented CTL). The differences with our work are several. From a process point of view, we focus not only on the verification but also on the validation of the generated models and on safety assessment; the outcome of our process is more informative since it relates the set of configurations with the properties that are satisfied (i.e., parameter synthesis). Finally, we integrate the modeling phase with a compositional approach that helps to save significant modeling effort. In principle, we could try to combine multiple configurations in order to analyze them together in a symbolic way. However, this was not needed and, on the contrary, the ability to work on each configuration independently made it possible to exploit high levels of parallelism provided by modern computing infrastructures.

7 Conclusions and Future Work

In this paper, we presented and released a complex real-world case study demonstrating the application of formal methods to the analysis of the big design space associated with the NextGen Automated Air Traffic Control System under study at NASA. Our approach resulted in a wealth of interesting data that supported the re-discovery of known results, and also the identification of new insights. When we started, NASA engineers had many possible design ideas, all described informally. We helped them to formalize and clarify these ideas, and to make explicit hidden assumptions. To the best of our knowledge, this is the first time that a design space of this scale has been mapped out by considering every possible solution in such depth.

The task of analyzing all the 1,620 designs would have been unfeasible without the novel process that we introduce. Our process combines and builds upon existing techniques and tools to perform model generation, validation, verification, and safety assessment. The process relies on a compositional, parametric, and contract-based approach in order to maximize reuse, and to ensure great confidence in the models by means of aggressive model validation. Overall, our study shows that this technology is mature and able to assist designers in formalizing and narrowing down design choices in an early phase of system design.

We extracted meaningful information from this data, and we expect that even more will be extracted in the future, working in collaboration with the NASA domain experts. In the future, we plan to extend the model by identifying additional modeling dimensions of interest, e.g., the fact that ADS-B information might not propagate equally to all aircraft, or the presence of multiple ATCs. Finally, we plan to leverage more the contract-based infrastructure defined in this work, in order to identify properties that can be proved by pure compositional reasoning. We believe that this process can be applied to other design exploration situations in which the size of the design space stems from the local variability of the components. For example, we have started working with the World Bank through Data Science for Social Good [1] to use an adaptation of the framework presented in this paper to help them root out corruption, collusion, and fraud by comparatively analyzing the temporal behaviors of their large network of suppliers.

References

1. Eric & Wendy Schmidt Data Science for Social Good, University of Chicago, `http://dssg.uchicago.edu/`
2. Nasa airspace operations and safety program, `http://www.aeronautics.nasa.gov/programs-aosp.htm`
3. Nasa nextgen-airspace, `http://www.hq.nasa.gov/office/aero/asp/airspace/`
4. Project webpage: Formal methods for automated airspace concepts, `https://es-static.fbk.eu/projects/nasa-aac`
5. NextGen (May, 2016), `https://www.faa.gov/nextgen/`

6. Abadi, M., Lamport, L.: Composing Specifications. ACM Transactions on Programming Languages and Systems 15(1), 73–132 (1993)
7. ARP4754A Guidelines for Development of Civil Aircraft and Systems. SAE (Dec 2010)
8. ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, SAE (Dec 1996)
9. Bauer, C., Lagadec, K., Bès, C., Mongeau, M.: Flight control system architecture optimization for fly-by-wire airliners. Journal of guidance, control, and dynamics 30(4), 1023–1029 (2007)
10. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP safety analysis platform. Proceedings of 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2016)
11. Bozzano, M., Cimatti, A., Griggio, A., Mattarei, C.: Efficient Anytime Techniques for Model-Based Safety Analysis. In: CAV (2015)
12. Bozzano, M., Cimatti, A., Mattarei, C.: Automated Analysis of Reliability Architectures. In: 18th International Conference on Engineering of Complex Computer Systems (ICECCS). pp. 198–207. IEEE (july 2013)
13. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on 100(8), 677–691 (1986)
14. Butler, R.W., Hagen, G., Maddalon, J.M.: The Chorus conflict and loss of separation resolution algorithms. Tech. rep., Technical Memorandum NASA/TM-2013-218030, NASA, Langley Research Center, Hampton VA 23681-2199, USA (2013)
15. Can, A.B., Bultan, T., Lindvall, M., Lux, B., Topp, S.: Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers. Automated Software Engineering 14(2), 129–178 (2007)
16. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 8559, pp. 334–342. Springer (2014)
17. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A Tool for Checking the Refinement of Temporal Contracts. In: ASE. pp. 702–705. IEEE (2013)
18. Cimatti, A., Tonetta, S.: Contracts-refinement proof system for component-based embedded systems. Science of Computer Programming 97, 333–348 (2015)
19. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 321–330. ACM (2011)
20. von Essen, C., Giannakopoulou, D.: Analyzing the next generation airborne collision avoidance system. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 620–635. Springer (2014)
21. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT-Workshop (2015)
22. Hagen, G., Butler, R., Maddalon, J.: Stratway: a modular approach to strategic conflict resolution. In: Preceedings of 11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, Virgina Beach, VA (2011)
23. Idris, H.R., Shen, N., Wing, D.J.: Improving separation assurance stability through trajectory flexibility preservation. In: 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference. p. 9011 (2010)
24. Jeannin, J.B., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., Platzer, A.: A formally verified hybrid system for the next-generation airborne

collision avoidance system. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 21–36. Springer (2015)
25. Karr, D.A., Vivona, R.A., Roscoe, D.A., DePascale, S.M., Wing, D.J.: Autonomous Operations Planner: A Flexible Platform for Research in Flight-Deck Support for Airborne Self-Separation. In: 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. p. 5417 (2012)
26. Lauderdale, T., Lewis, T., Prevot, T., Ballin, M., Aweiss, A., Guerreiro, N.: Function allocation for separation assurance: Research plan (Aug 2014), NASA HQ Project Overview
27. Loos, S.M., Renshaw, D., Platzer, A.: Formal verification of distributed aircraft controllers. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control. pp. 125–130. HSCC '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2461328.2461350
28. Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., Kristin Yvonne, R.: Comparing different functional allocations in automated air traffic control design. In: Formal Methods in Computer-Aided Design (FMCAD15) (2015)
29. Mehlitz, P.: Trust your model-verifying aerospace system models with Java PathFinder. IEEE/Aero (2008)
30. Munoz, C., Carreño, V., Dowek, G.: Formal analysis of the Operational Concept for the Small Aircraft Transportation System. In: Rigorous Development of Complex Fault-Tolerant Systems, pp. 306–325. Springer (2006)
31. Muñoz, C., Siminiceanu, R., Carreño, V., Dowek, G.: KB3D reference manual-version 1. NASA (2005)
32. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook. Tech. Rep. NUREG-0492, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. (1981)
33. Wing, D.J., Ballin, M.G., Krishnamurthy, K.: Pilot in command: a feasibility assessment of autonomous flight management operations. In: 24th International Congress of the Aeronautical Sciences (2004)
34. Zhao, Y., Rozier, K.Y.: Formal specification and verification of a coordination protocol for an automated air traffic control system. Science of Computer Programming Journal 96(3), 337–353 (December 2014)
35. Zhao, Y., Rozier, K.Y.: Probabilistic model checking for comparative analysis of automated air traffic control systems. In: Proceedings of the 33rd IEEE/ACM International Conference On Computer-Aided Design (ICCAD 2014). pp. 690–695. IEEE/ACM, San Jose, California, U.S.A. (November 2014)