Probabilistic Model Checking for Comparative Analysis of Automated Air Traffic Control Systems

Yang Zhao
Microsoft, Redmond, WA 98052, USA
Email: yanzhao@microsoft.com

Kristin Y. Rozier
NASA Ames Research Center, Moffett Field, CA 94035, USA
Email: Kristin.Y.Rozier@nasa.gov

***Abstract*—Ensuring aircraft stay safely separated is the primary consideration in air traffic control. To achieve the required level of assurance for this safety-critical application, the Automated Airspace Concept (AAC) proposes a network of components providing multiple levels of separation assurance, including conflict detection and resolution. In our previous work, we conducted a formal study of this concept including specification, validation, and verification utilizing the NuSMV and CadenceSMV model checkers to ensure there are no potentially catastrophic design flaws remaining in the AAC design before the next stage of production. In this paper, we extend that work to include probabilistic model checking of the AAC system.[1] We are motivated by the system designers requirement to compare different design options to optimize the functional allocation of the AAC components. Probabilistic model checking provides quantitative measures for evaluating different design options, helping system designers to understand the impact of parameters in the model on a given critical safety requirement. We detail our approach to modeling and probabilistically analyzing this complex system consisting of a real-time algorithm, a logic protocol, and human factors. We utilize both Discrete Time Markov Chain (DTMC) and Continuous Time Markov Chain (CTMC) models to capture the important behaviors in the AAC components. The separation assurance algorithms, which are defined over specific time ranges, are modeled using a DTMC. The emergence of conflicts in an airspace sector and the reaction times of pilots, which can be simplified as Markov processes on continuous time, are modeled as a CTMC. Utilizing these two models, we calculate the probability of an unresolved conflict as a measure of safety and compare multiple design options.**

I. Introduction

Safe separation of aircraft is the top priority in the Next Generation Air Transportation System (NextGen) Automated Airspace Concept (AAC) [11]. To achieve the required level of assurance for this safety-critical application, the AAC proposes a network of components providing multiple levels of fault-tolerance. The AutoResolver [10] works with the human air traffic controller to provide strategic separation assurance, including rerouting aircraft to avoid a Loss of Separation (LoS) event projected to occur between about 20 minutes to about three minutes out. Complimenting the AutoResolver and controller, the Tactical Separation Assurance Flight Environment (TSAFE) [20] provides tactical separation assurance, resolving LoS events projected to occur from about three minutes out to about 30 seconds out. These systems follow rigorous protocols for transfer of control of aircraft, for example enabling the controller to choose to control an aircraft or transfer that control to TSAFE. There is also a fail-safe TSAFE threshold where TSAFE will automatically take control of an aircraft in the presence of an otherwise unhandled predicted LoS projected to occur one minute in the future [11]. The operational concept defines protocols for control hand-offs both before and after a projected LoS. On-board, the pilots follow protocols including adhering to either today's TCAS or the NextGen Airborne Collision Avoidance System (ACAS X) [23] systems for conflict detection and resolution.

In our previous work [24], [25], we conducted a formal study of the AAC system-level logic design, including specification, validation, and verification. We created SMV models at different levels of abstraction and utilized model checking for system verification via both NuSMV and CadenceSMV. Specifically, we answered the fundamental design logic question: if every component operates perfectly 100% of the time, does the system satisfy its requirements? If not, either the system or the requirements need to be changed. It is pointless to test the system, e.g. with injected faults, if the system didn't work as expected in the presence of no faults. Our previous study found two important counterexamples that pinpointed unexpected emergent behaviors in the initial AAC design. Once the AAC system designers had updated the AAC design to eliminate these behaviors, we proved there are no potentially catastrophic design flaws remaining in the AAC system-level logic design before the next stage of production.

Now that we have proved at least one system configuration is safe in that it satisfies all of the designers' safety requirements [24], the most pressing problem facing system designers is that of *functional allocation*, or how to arrange the components of the AAC architecture in the best way. For example, we can make different hardware choices with different reliability characteristics, we can place either AutoResolver or TSAFE either on the ground or in the air and connect them in different ways, and there can be varying levels of redundancy in the system. (In this paper, we focus on the connection options between AutoResolver and TSAFE.) Finding any cases where one configuration is safer than another is vital for designing the safest system. Finding that two or more configurations are equivalent is also important. Safety is the first priority so establishing a set of logically equally safe configurations allows system designers to optimize for other pressing concerns, such as cost, fuel efficiency, minimizing flight delays, minimizing environmental impact factors, usability, etc.

In order to gather data to help evaluate the functional allocation question, NASA commissioned a study to determine the failure probabilities of both potential hardware and software components for the AAC [4]. We utilize this data

[1]All models, specifications, and run scripts are available at http://research.kristinrozier.com/ICCAD14.html. Work contributing to this paper was supported in part by NASA's Airspace Systems Program.

to extend our previous work [25] to include probabilistic model checking of the AAC system, consisting of a real-time algorithm, a logic protocol, and human factors. We detail our approach to high-level system logic modeling and discuss validation and system-level specification. Our models integrate hardware and software, including probabilities of failure for potential hardware choices of components to enable the system designers to decide the best functional allocation for their air traffic control system so they can design for resilience and robustness. The separation assurance algorithms, which are defined over specific time ranges, are modeled using a Discrete Time Markov Chain (DTMC). The emergence of conflicts in an airspace sector and the reaction times of pilots, which can be simplified as Markov processes on continuous time, are modeled as a Continuous Time Markov Chain (CTMC). We check the DTMC model against PCTL [14] properties and the CTMC model against CSL [3] properties. Utilizing these two models, we calculate the probability of an unresolved conflict; probabilistic model checking provides quantitative measures for evaluating different design options, helping system designers to understand the impact of parameters in the model on a given critical safety requirement.

A. Related Work

1) Probabilistic model checking: To our knowledge, our modeling method is new but the utility of probabilistic model checking for analyzing systems of communicating processes is well-known; [9] details such an analysis using probabilistic timed automata. We use PRISM as it is the most well-documented [16] probabilistic model checker that supports our probabilistic models. PRISM mostly uses decision diagram representations of state sets and transition matrices, which often results in better scalability for large models. MRMC [15] also supports the verification of PCTL and CSL against both DTMC and CTMC models. SMART [8] has comparable capabilities but lacks PRISM's expressive modeling language and DTMC support since it is more specialized for Petri net models from which it generates CTMCs. Other probabilistic model checkers including LiQuor [2] and PASS [13] do not provide our required features. The COMPASS project (Correctness, Modeling, and Performance of Aerospace Systems) [5], [6] introduces a model-based methodology for the design of aerospace systems integrating the modeling language AADL (Architecture Analysis and Design Language) with safety guarantees, performance, and dependability analysis.

2) The Automated Airspace Concept: All AAC components that calculate resolution maneuvers have undergone verification through simulation as well as ongoing component-level verification utilizing formal methods [19], [12], [7], [18], [23]. The fault-tolerance of the AutoResolver algorithm given trajectory prediction errors was evaluated via simulation in [17]. Other forms of safety analysis for the AAC were studied in previous papers [1], [4], [22], [21], all of which focus on the impact of hardware failures on the safety of the AAC. In [1], system safety is defined as a combination of basic events and modeled using fault trees. Also [21] uses a data structure called a dynamic event tree to capture the time-dependent behaviors in the AAC. Next [4], [22] considers a single pair of aircraft involved in the projected conflict and simulates scenarios when there are 8 minutes before projected collisions. It captures some dynamic properties of the AAC system, but does not consider the interaction between the controller/AutoResolver, TSAFE, and pilots defined by the operational concept, particularly when there are multiple conflicts detected. We also enhance their analysis by modeling the response delay from the controller and pilots. Regarding the analysis method, [1], [21] adopts an analytical approach, while [4], [22] employs Monte Carlo simulation. The AutoResolver algorithm was recently updated to better coordinate multiple parallel AutoResolver instances controlling different FAA Air Route Traffic Control Centers [18], and evaluated using simulation. In this paper, we model the AAC system as a Markov chain and analyze it using the probabilistic model checker PRISM, thereby studying the full impact of the complete set of system behaviors defined by the operational concept on the safety of the AAC system, which has never been investigated previously. In other words, we not only consider hardware failure scenarios, but also the dynamic execution of the complete AAC, which includes hardware, software, and human factors, according to its definition in the operational concept. All our experiments were carried out using PRISM version 4.1.

The remaining paper is structured as follows: Section II details our system models and modeling strategy along with the temporal logic specifications we write for probabilistic model checking. We probabilistically analyze candidate configurations for the AAC at the system level, demonstrating when LoS is best prevented, in Section III. Section IV concludes and points to the next directions for design-time analysis of the AAC.

II. Modeling the AAC in PRISM

There are two stages of LoS resolution: the process of conflict detection and then resolution generation, and the process of resolution transmission and execution. To successfully resolve a potential LoS, either TSAFE or AutoResolver should first detect it and calculate resolution maneuvers. TSAFE and AutoResolver are designed to independently address conflicts that occur in different time ranges. As we show in Section II-A, the behaviors of TSAFE and AutoResolver for separation assurance can be defined over discrete time values. We use a DTMC model to analyze the separation assurance processes of AutoResolver and TSAFE, and the time to conflict is modeled as a state variable. The above problem is then reduced to a probabilistic reachability problem in the DTMC.

The second stage involves the coordination of TSAFE and AutoResolver defined in the operational concept in the AAC. When both AutoResolver and TSAFE attempt to resolve a LoS, a protocol would be followed based on predicted time to the LoS. We also need to consider the human behaviors including pilots' and controllers' behaviors. We build a CTMC model to address this factor. Based on the calculated probabilities from the DTMC model, we have rates for AutoResolver/TSAFE alerting and giving resolutions to the controller or pilots.

We are primarily interested in the probability of LoS. There are two factors that contribute to this measure: the first is that both AutoResolver and TSAFE fail to detect and resolve the LoS (which is extremely unlikely given the verification research described in Section I-A2); the second is that either or both AutoResolver and TSAFE successfully generate the resolutions, but the resolutions are not executed on time. The second case requires more analysis and is the focus of the remainder of this paper.

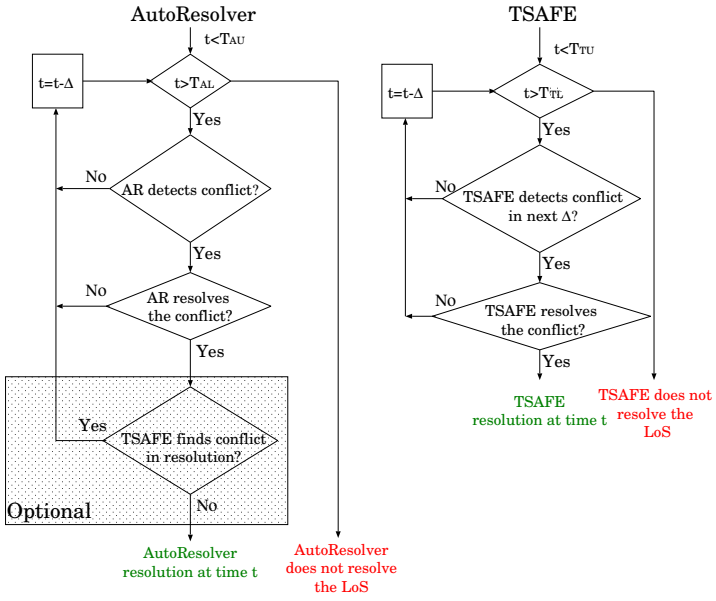

Fig. 1. Flow Diagram for the interaction of AutoResolver and TSAFE, where t is the time remaining until the projected conflict.

We introduce the term *resolution scenario*. The resolution scenario (RS) is defined as a pair $\langle t_1,t_2\rangle$, where t_1 is the time that AutoResolver generates resolution before a projected LoS, and t_2 is the time for TSAFE. If AutoResolver or TSAFE fail to generate a resolution, the corresponding field will be denoted as $*$. Based on system behavior, only a finite set of resolution scenarios are possible and there is a probability distribution on this set of possible resolution scenarios; the probability of an RS $\langle t_1,t_2\rangle$ is denoted by $P_{res}(\langle t_1,t_2\rangle)$. For RS $\langle t_1,t_2\rangle$, we denote the probability that neither resolution (if generated) is executed on time by $P_{noexe}(\langle t_1,t_2\rangle)$.

To summarize, the probability P_{LoS} that LoS still occurs can be calculated by:

$$P_{LoS} = P_{nores} + \sum_{\langle t_1,t_2\rangle} P_{res}(\langle t_1,t_2\rangle) \cdot P_{noexe}(\langle t_1,t_2\rangle). \quad (1)$$

The DTMC model is devoted to generating all resolution scenarios and their probabilities respectively, and the CTMC model computes the probability P_{noexe} for each RS.

A. DTMC model: LoS detection by AutoResolver and TSAFE

By modeling the separation assurance processes of AutoResolver and TSAFE, a distribution of time to conflict when AutoResolver and/or TSAFE generate resolutions can be obtained. We build a DTMC to calculate the probability distribution among RSs. In other words, we need to answer the question: what is the probability that AutoResolver generates resolution at t_1 (ahead of the LoS) and TSAFE does that at t_2 for a potential LoS?

We first introduce the process of AutoResolver and TSAFE LoS detection and resolution. Both of these two components rely on radars to obtain the current trajectories for all aircraft, and radars scan air section periodically with a fixed frequency.

In each air sector, AutoResolver and TSAFE execute in parallel to tackle LoSs in different time ranges. AutoResolver is designed for LoSs between T_{AU} (upper-bound time for AutoResolver) and T_{AL} (lower-bound time for AutoResolver) ahead of projected time of LoS, and TSAFE for those between T_{TU} (upper-bound time for TSAFE) and T_{TL} (lower-bound time for TSAFE) ahead of projected time of LoS. Hence, in a RS $\langle t_1,t_2\rangle$, $t_1 \in [T_{AL}, T_{AU}]$ if $t_1 \neq *$ and $t_2 \in [T_{TL}, T_{TU}]$ if $t_2 \neq *$. Both AutoResolver and TSAFE follow a loop procedure, and each loop corresponds to a radar period. Fig. 1 shows the procedures of detecting and resolving a LoS for AutoResolver and TSAFE respectively, and Fig. 2 demonstrates the corresponding timeline for AutoResolver. The basic separation assurance process can be described as follows:

- AutoResolver checks for LoS every radar cycle Δ_{Ad}, called the AutoResolver detection interval. If no conflict is detected during a check, AutoResolver will retry after Δ_{Ad}.
- The probability that AutoResolver and TSAFE detect that the conflict could occur in time t is given by a formula $p(t)$.
- If AutoResolver detects and successfully resolves a LoS in future, AutoResolver generates a resolution, and the resolution time is current time t. If the conflict is not resolved by the next Δ_{Ad}, i.e. the trajectories of the involved aircraft are still projected to lead to LoS, then it will retry resolution in next detection process.

The TSAFE separation assurance process follows a similar detection and then resolution routine with a different radar cycle Δ_{Td}. In both TSAFE and AutoResolver, the detection probability $p(t)$ is given by [4]:

$$p(t) = (-6.2 \cdot 10^{-12})t^4 + (1.2 \cdot 10^{-8})t^3 + (-6.3 \cdot 10^{-6})t^2 + (-3.0 \cdot 10^{-4})t + 1.0$$

A practical issue to be addressed in the above process is the possibility that a TSAFE conflict may be caused by an AutoResolver resolution. Since AutoResolver and TSAFE use different detection and resolution algorithms based on different sets of information, a resolution generated by AutoResolver may immediately trigger a TSAFE conflict. A proposed fix to this issue is to check all AutoResolver resolutions using TSAFE: after AutoResolver generates a resolution, this resolution will be given to TSAFE before being sent to the aircraft involved. If this resolution causes a TSAFE conflict, the AutoResolver would consider this round of detection and resolution to have failed and, rather than sending the proposed resolution to the aircraft involved, AutoResolver will instead retry in the next round. The step of adding a TSAFE check, depicted in the shaded square in Fig.1, brings both positive and negative effects to the system: it reduces the probability of a TSAFE conflict but it requires more time to resolve an AutoResolver conflict. An important motivation for our experiments is to quantify how this potential new design affects the reliability of the overall AAC system. Our analysis is detailed in Section III.

We use two Boolean resolution variables: AR_res indicates whether AutoResolver has generated a resolution and $TSAFE\_res$ indicates whether TSAFE has generated a resolution. Another two variables, AR_res_time and $TSAFE\_res\_time$, record the times these resolutions were generated, which is particularly important if corresponding resolutions are generated for the same conflict. There are four

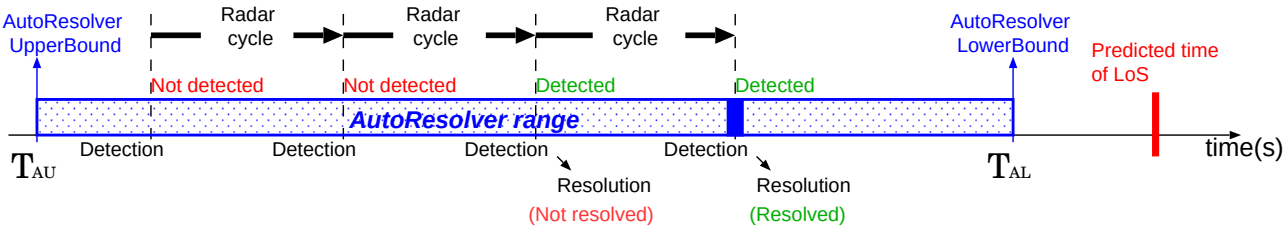

Fig. 2. AutoResolver timeline for LoS detection and resolution

types of terminal states, and the only outgoing transition from the a terminal state is a self-loop:

- $AR\_res = 1$ and $TSAFE\_res = 1$, in this case RS is $\langle AR\_res\_time, TSAFE\_res\_time \rangle$.
- $AR\_res = 1$ and $TSAFE\_res = 0$, in this case RS is $\langle AR\_res\_time, * \rangle$.
- $AR\_res = 0$ and $TSAFE\_res = 1$, in this case RS is $\langle *, TSAFE\_res\_time \rangle$.
- $AR\_res = 0$ and $TSAFE\_res = 0$, in this case no resolution is generated.

We use the following PCTL formula to check against the DTMC model:

$$S_{=?}[TSAFE\_res = 1 | AR\_res = 1]$$

which queries the steady-state probability that either AutoResolver or TSAFE generate resolutions for the LoS. With the verbose option "-v", PRISM will report the probability of reaching all bottom strongly-connected components (BSCCs). We use a python script to check the value of AR_res, $TSAFE\_res$, AR_res_time, and AR_res_time. All possible RSs and their probabilities can be obtained from the PRISM output.

B. CTMC model: the complete AAC system

We use a CTMC model to analyze the process that generated resolutions are transmitted to involved aircraft. The system behaviors are modeled as a CTMC. For simplicity, we assume the response time for pilots follows the exponential distribution. The structure of our CTMC model is shown in Fig.3. As shown in the above subsection, the distribution of RS, which is obtained from DTMC, as the input. The objective of the analysis on the CTMC model is to compute the probability $P_{nonexe}(\langle t_1, t_2 \rangle)$ for each $\langle t_1, t_2 \rangle$.

In the CTMC model, we use the variables AR_res and $TSAFE\_res$ to indicate whether there are AutoResolver or TSAFE resolutions respectively. Moreover, $TSAFE\_res = BT$ means the TSAFE resolution is before the TSAFE threshold, and $TSAFE\_res = AT$ means that it is after the TSAFE threshold. We model two aircraft in the CTMC model. Each aircraft maintains two Boolean variables, AR_cmd and $TSAFE\_cmd$, that indicate whether the aircraft has received a resolution from AutoResolver or TSAFE, respectively. According to the resolution priority protocol defined in the AAC operational concept [11], if both TSAFE and AutoResolver resolutions are received, the pilot will always execute the TSAFE resolution.

The CTMC model enjoys a "memoryless" property that the system behaviors starting from a certain time only depend

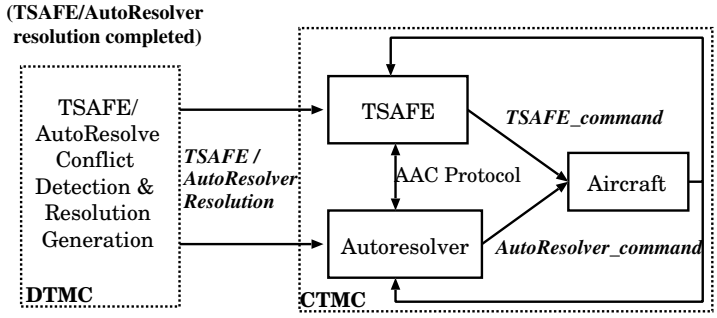

Fig. 3. CTMC module

on the state at that time. Thus, it is safe to divide the system execution path into several segments and analyze them independently. Our approach is to calculate the probability of each segment with different initial states separately. There are four initial settings we need to consider:

1) There is an AutoResolver resolution and no TSAFE resolution, and the time is before the TSAFE threshold. In the initial state:
$$AR\_res = 1; \ TSAFE\_res = 0; \quad (2)$$
2) There are both AutoResolver and TSAFE resolutions, and the time is before the TSAFE threshold. In the initial state:
$$AR\_res = 1; \ TSAFE\_res = BT; \quad (3)$$
3) There is only a TSAFE resolution, and the time is before the TSAFE threshold. In the initial state:
$$AR\_res = 0; \ TSAFE\_res = BT; \quad (4)$$
4) There is only a TSAFE resolution, and the time is after the TSAFE threshold. In the initial state:
$$AR\_res = 0; \ TSAFE\_res = AT; \quad (5)$$

We use a Boolean variable *conflict* to denote whether there is an unresolved LoS in the future. In all the above initial state, $conflict = 1$. When the LoS is successfully resolved, this variable will be reset to 0.

We use an example to show how to calculate $P_{nonexe}(\langle AR\_res\_time, TSAFE\_res\_time \rangle)$. Fig. 4 depicts the timeline for all resolution scenarios. If the LoS is detected and resolved by AutoResolver and TSAFE, but the resolution is not successfully executed on time, it means the following sequence of behaviors occurs:

1) Between AR_res_time to $TSAFE\_res\_time$ ahead of LoS, there is only an AutoResolver resolution, and this resolution is not executed within time $T = AR\_res\_time - TSAFE\_res\_time$. The CTMC model starts from the initial state in Formula 2.
2) In the range from $TSAFE\_res\_time$ to T_{AL} ahead of LoS, where T_{AL} is the AutoResolver lower bound,

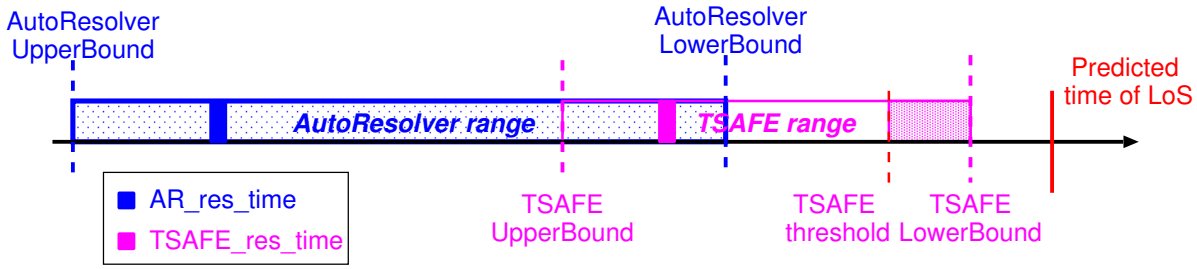

Fig. 4. AutoResolver and TSAFE resolution scenario timeline

there are both AutoResolver and TSAFE resolutions. These two resolutions are not executed within time $T = TSAFE\_res\_time - T_{AL}$. The CTMC model starts from the initial state shown in Formula 3.

3) In the range from T_{AL} to T_{TThres}) ahead of LoS, where T_{TThres} is the TSAFE threshold, there is only a TSAFE resolution and it is before the TSAFE threshold. This resolution is not executed within time $T = T_{AL} - T_{TThres}$. The CTMC model starts from the initial state shown in Formula 4.
4) In the range from T_{TThres} to T_{TL}) ahead of LoS, where T_{TL} is the TSAFE lower bound, there is only a TSAFE resolution and it is after the TSAFE threshold. This resolution is not executed within time $T = T_{TThres} - T_{TL}$. The CTMC model starts from the initial state shown in Formula 5.

The probabilities for the above four cases can be obtained by checking the model with the specific initial state against the CSL property:

$$P_{=?}[conflict != 0 \ \mathsf{U}^{>T} \ conflict = 0],$$

where T is given in each above stage. This formula queries the probability that the existing LoS is not resolved during the time interval T.

Validation: We carried out basic model validation manually. PRISM is able to dump state space and quantitative information. For DTMCs, states in each BSCC and its probabilistic reachability can be dumped out. For CTMCs, the size of transition matrix and maximum diagonal value can be printed. Many errors introduced in the modeling phase can be found by manually checking on this information against the operational concept.

III. System-Level Probabilistic Comparison of AAC Design Candidates

The main measure we investigate is the probability of LoS, which can be calculated based on the previous discussion, and the relationship between this probability and other parameters. Since AutoResolver and TSAFE are two different algorithms, and they try to resolve the conflict based on different sets of information and different levels, it is possible that a resolution generated by AutoResolver may result in a TSAFE conflict. We consider two different design options.

Checking scheme: We can make TSAFE detection an integral part of the AutoResolver conflict detection cycle, so that TSAFE checks whether any AutoResolver-generated resolutions will result in a TSAFE conflict. If so, AutoResolver will discard this resolution and try to detect and resolve again in the next round. If not, AutoResolver will send the resolution as usual. Fig. 1 shows this scheme *with* the optional part.

Scheme	P_{nores}
Checking	$2.22e-16$
No checking	$2.22e-16$

TABLE I. Probability that both AutoResolver and TSAFE fail to detect and resolve the LoS.

No-checking scheme: AutoResolver directly sends its resolution without checking with TSAFE. There is probability p_{in} a TSAFE conflict will occur after this resolution is executed. Fig. 1 shows this scheme *without* the optional part.

We first consider the probability P_{nores} that neither AutoResolver nor TSAFE successfully resolves the potential LoS. To accomplish this, we check the following PCTL property against our DTMC model:

$$P_{=?}[\mathsf{F} \ AR\_res = 0 \ \& \ TSAFE\_res = 0]$$

Tab. I shows the probabilities for both schemes, the results are almost the same. As we show later, $P_{noexe} \ll \sum_{\langle t_1,t_2\rangle} P_{res}(\langle t_1,t_2\rangle) \cdot P_{noexe}(\langle t_1,t_2\rangle$, meaning the probability that LoS is not detected and resolved by AutoResolver or TSAFE is much less than the probability that the resolutions are not executed on time. We can only focus on the right side of the above equation and ignore the probability P_{noexe}.

The resulting probability of LoS is a primary consideration for designers to make a decision between these two schemes. There is a trade-off between the delay that AutoResolver generates the resolution and the save from eliminating the potential TSAFE conflict caused by AutoResolver. The probability that AutoResolver resolution causes a TSAFE conflict, denoted by P_c, plays an important role in this decision making. This parameter is an input to our model and should be obtained from analyzing the AutoResolver and TSAFE algorithms and also statistical results from the practical data. Intuitively, when P_c is "very small," the no-checking scheme seems to be a better choice, while the checking scheme tends to outperform the no-checking scheme when P_c is large. We need to obtain an quantitative understanding about the impact of P_c to the decision we make between these two schemes.

We plot the probabilities of LoS for these two schemes under different P_c in Fig. 5, where the x-axis represents the P_c and the y-axis represents the probability of LoS. We can clearly see that the results of the checking scheme are not sensitive to P_c while those of the no-checking scheme are much more. There is a cross of these two curves when $P_c = P'c \approx 0.015$. When $P_c < P'c$ the no-checking scheme results in a smaller probability of LoS, and the checking scheme is better when $P_c > P'c$. Based on this result, system designers have a clearer understanding of how to choose between these two schemes under different parameters P_c.

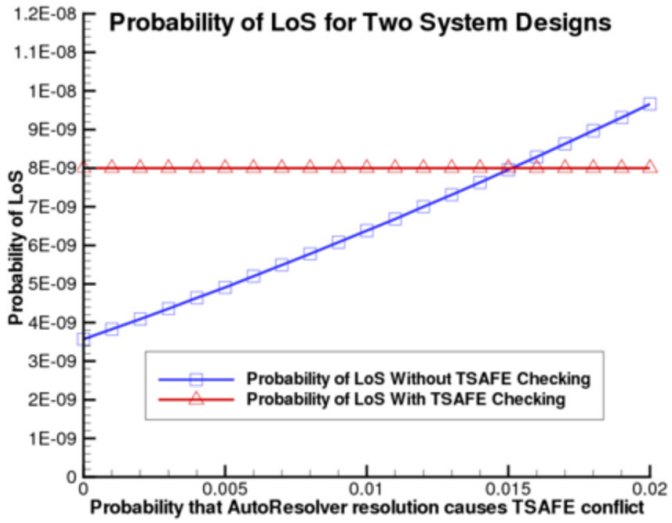

Fig. 5. Probabilities of LoS with P_c. We check models with (checking scheme) and without (no-checking scheme) the optional step in Fig. 1 using different probability P_c and calculate the probability of LoS using Formula 1. The scheme resulting in lower probability of LoS is more preferable.

IV. Conclusions and Future Work

While the important topic of AAC safety analysis has necessitated many different studies from different angles, ours is the first to take into account the full impact of the complete set of system behaviors defined by the operational concept on the safety of the AAC at a system level. In this paper, we detail system-level specification, modeling, and analysis, focusing on the comparison of two candidate designs. Utilizing probabilistic model checking to check temporal properties of the system, we can provide the system designers with valuable information regarding which design to choose, depending on the probability that any given AutoResolver resolution could cause a TSAFE conflict. When the designs for AutoResolver and TSAFE resolution generation and possible integration are complete, the choice of whether to run them independently or in tandem will take into account our work.

Validation for probabilistic models like those we present in this paper is hard. As we expand and update our models for continued design-time verification and functional allocation analysis of the AAC we look to develop better methods for quantitative validation of both our models and specifications. We would also like to explore methods for specification debugging and other sanity checks for the logics PCTL and CSL. All of these measures would help us expand our analysis in a robust way and serve to increase confidence in the impact of our findings as the AAC system design continues to evolve and mature and our models necessarily grow ever more complex.

References

[1] J. Andrews, Welch J., and Erzberger H. Safety analysis for advanced separation concepts. *Air Traffic Control Quarterly*, 14(1), 2006.

[2] C. Baier, F. Ciesinski, and M. Größer. Probmela and verification of markov decision processes. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):22–27, 2005.

[3] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV*, pages 358–372, 2000.

[4] D. Blum, D. Thipphavong, T. Rentas, Y. He, X. Wang, and M.E. Patẽ-Cornell. Safety analysis of the advanced airspace concept using Monte Carlo simulation. *AIAA Meeting Papers on Disc*, 15(9), 2010.

[5] M. Bozzano, A. Cimatti, J. Katoen, V. Ngsuyen, T. Noll, and M. Roveri. The COMPASS approach: correctness, modelling and performability of aerospace systems. In *SAFECOMP, 5775, LNCS*, pages 173–186. Springer, 2009.

[6] M. Bozzano, A. Cimatti, J-P. Katoen, V. Nguyen, T. Noll, M. Roveri, and R. Wimmer. A Model Checker for AADL. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 562–565. Springer, 2010.

[7] D. Bushnell, D. Giannakopoulou, P. Mehlitz, R. Paielli, and C. Pasareanu. Verification and validation of air traffic systems: Tactical separation assurance. In *IEEE Aerospace Conf.*, pages 1 –10, 2009.

[8] G. Ciardo, A.S. Miner, M.-Y. Chung, R.L. Jones, R. M. Marmorstein, R. I. Siminiceanu, and A. Yu. SMART: Stochastic Model checking Analyzer for Reliability and Timing, User Manual, 2006. Available at http://www.cs.ucr.edu/ ciardo/SMART/SMARTman.pdf.

[9] M. Duflot, M. Kwiatkowska, G. Norman, D. Parker, S. Peyronnet, C. Picaronny, and J. Sproston. *Formal Methods for Industrial Critical Systems: A Survey of Applications*, chapter 7: Practical Applications of Probabilistic Model Checking to Communication Protocols, pages 133–150. John Wiley & Sons, Ltd. and IEEE Comp. Soc. Press, 2013.

[10] H. Erzberger. Separation assurance in the future air traffic system. In *Proceedings of the ENRI International Workshop on ATM/CNS*, 2009.

[11] H Erzberger and K Heere. Algorithm and operational concept for resolving short-range conflicts. *Proc. IMechE G J. Aerosp. Eng.*, 224(2):225–243, 2010.

[12] D. Giannakopoulou, D. Bushnell, J. Schumann, H. Erzberger, and K. Heere. Formal testing for separation assurance. *AMAI*, 63:5–30, 2011.

[13] E.M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. Pass: Abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357. Springer, 2010.

[14] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[15] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *QEST*, pages 243–244. IEEE Computer Society, 2005.

[16] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV*, LNCS 6806, pages 585–591. Springer, 2011.

[17] T.A. Lauderdale, A.C. Cone, and A.R. Bowe. Relative significance of trajectory prediction errors on an automated separation assurance algorithm. In *9th USA/Europe ATM R&D Seminar (ATM2011)*, 2011.

[18] T.A. Lauderdale and T. Wang. Coordination between multiple ground-based separation assurance agents. In *AIAA Aviation Technology, Integration, and Operations Conference*, 2013.

[19] R. Paielli, H. Erzberger, D. Chiu, and K. Heere. Tactical conflict alerting aid for air traffic controllers. *J Guid Contr Dynam*, 32(1):184–193, 2009.

[20] R.A. Paielli. Tactical conflict resolution using vertical maneuvers in enroute airspace. *AIAA Journal of Aircraft*, 45(6), Nov-Dec 2008.

[21] J. Shortle, L. Sherry, A. Yousefi, and R. Xie. Safety and sensitivity analysis of the advanced airspace concept for nextgen. In *Integrated Communications, Navigation and Surveillance Conference (ICNS), 2012*, pages O2–1 –O2–10, april 2012.

[22] D. Thipphavong. Accelerated Monte Carlo simulation for safety analysis of the advanced airspace concept. In *AIAA ATIO/ISSMO Conference*, 2010.

[23] C. von Essen and D. Giannakopoulou. Analyzing the next generation airborne collision avoidance system. In *TACAS*, pages 620–635. Springer, 2014.

[24] Y. Zhao and K.Y. Rozier. Formal specification and verification of a coordination protocol for an automated air traffic control system. In *Proc. AVoCS*, volume 53 of *Electronic Communications of the EASST*. European Association of Software Science and Technology, 2012.

[25] Y. Zhao and K.Y. Rozier. Formal specification and verification of a coordination protocol for an automated air traffic control system. *Science of Computer Programming Journal*, 2014.