An Approach to the Verification of the Center-TRACON Automation System*

John Lygeros, George J. Pappas and Shankar Sastry

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley CA 94720
lygeros,gpappas,sastry@eecs.berkeley.edu

Abstract. The Center-TRACON Automation System (CTAS) is a collection of planning and control software functions that generate landing schedules and advisories to assist air traffic controllers in handling traffic in the en-route and terminal areas. In this paper, we propose a formal safety analysis methodology to determine the correctness of CTAS with respect to safety. Four large classes of safety notions are identified for the CTAS problem: nominal, robust, structural and degraded. For nominal safety questions we seek conditions under which the system is guaranteed to be nominally safe.

1 Introduction

The increasing demand for air travel has spurred the development of tools to increase airspace utilization, smooth air traffic flow and reduce fuel consumption, time delays and controller workload. In an effort to meet these objectives, NASA has developed the Center-TRACON Automation System (CTAS) [1]. CTAS is a collection of planning and control functions which generate advisories to assist, but not replace, the controllers in handling traffic in the Center and TRACON areas.

The structure and functionality of CTAS are briefly discussed in Section 2. CTAS is a large scale, safety critical software system, that should ideally be validated before it is deployed. The validation process is complicated by the fact that the overall system is hybrid, as the (primarily) discrete dynamics of the algorithm are coupled with the continuous dynamics of the aircraft and the human operators. We present a formal approach to the safety analysis of the CTAS system. We view our work as a first step towards the development of a general methodology for the analysis of large scale, hybrid software systems[2]. Our methodology proceeds in the following steps:

* Research supported by the FAA and NASA under Research Contract DTFA03-97-D-0004 and Grant 96-C-001 and by the Army Research Office under Grant DAAH 04-95-1-0588.

[2] For another example of such a system in air traffic control see [2].

- **System Modeling**: The CTAS system is modeled in the *Hybrid Input-Output Automata* formalism (Section 3).
- **Safety Specification**: We identify notions of safety and determine the desired system specification. We consider four classes of safety measures: *nominal, robust, structural* and *degraded safety* (Section 4).
- **Safety Analysis**: Given the CTAS model in the hybrid input-output automata formalism and the nominal safety specification, deductive techniques will be used to determine conditions under which the system satisfies the specification. To tackle the complexity of the analysis, high level specifications (at the level of CTAS) are partitioned to sub specifications for the lower level components. The components are analyzed individually to determine whether they meet the corresponding specifications. The proof for the overall CTAS system is composed from the component proofs using abstraction relations.

2 CTAS Overview

The Air Traffic Control system consists of three types of control facilities: Air Route Traffic Control Centers (Centers) which control en-route flights, Terminal Radar and Approach Control facilities (TRACON) which control arriving and departing flights within 30 nautical miles of airports and Airport Control Towers which control traffic in the immediate vicinity of the airport and on the ground. In the United States there are 20 Centers and over 400 TRACONs. The Center-TRACON Automation System provides advisories for the air traffic controllers, in an attempt to increase airspace utilization, reduce delays, fuel consumption and controller workload and improve safety. CTAS consists of three main components:

- **Traffic Management Advisor (TMA)**
- **Descent Advisor (DA)**
- **Final Approach Spacing Tool (FAST)**

TMA [3] and DA [4] coexist and operate in Center airspace whereas FAST [5] operates as a stand alone in TRACON airspace. Even though currently CTAS deals only with arrival traffic, future versions will incorporate the User Preferred Routing tool (UPR) [6] for en route traffic to support Free Flight [7] and the Expedited Departure Path tool (EDP) for departure traffic. UPR and EDP will coexist with TMA and DA in the Center Airspace whereas FAST will be in charge of all terminal area traffic. Currently, stand alone versions of TMA and FAST are being field tested at Dallas-Fort Worth whereas DA is being field tested at Denver.

2.1 CTAS Architecture

The architecture of CTAS is shown in Figure 1. CTAS is a human centered control system. It receives information from the aircraft and computes schedules

and advisories, which are then transmitted to the aircraft by the controllers. The feedback nature of the architecture makes CTAS reactive. If aircraft do not follow the advisories or controllers manually change the landing schedule, CTAS will readjust and produce new advisories in the next computation cycle. Thought of as a large input-output system, CTAS receives input from:

- **Controllers:** The Traffic Management Coordinator (TMC), who resides in a Center, sets the capacity and acceptance rates for various runways, airports and the TRACON and can alter the landing sequence or schedule. The Center and TRACON controllers may select particular routes or runways for particular aircraft and impose constraints on the landing sequence or routing. Controller preferences are inputed through graphical user interfaces: the TMA Graphical User Interface (TGUI), which is used by the TMC, and the Plainview Graphical User Interface (PGUI), which is used by all Center and TRACON controllers.
- **Radar Daemon:** The radar daemon periodically receives aircraft state information regarding position, altitude, speed, aircraft type and flight plan.
- **Weather Daemon:** The weather daemon receives weather information from the National Weather Service. Currently the weather reports include wind, temperature and pressure profiles in the form of a three dimensional grid whose edge length is 50 miles. The weather reports are updated every hour and contain forecasts for the next three hours.

Internally, CTAS utilizes detailed databases which include aircraft, aerodynamic and engine models for all aircraft types. These models are used to perform accurate trajectory prediction for each aircraft. Other databases contain information on the local airspace structure in terms of way-points and routes, site adaptation data such as TRACON, airport and runway configurations and site specific constraints. A Communication Manager supports the internal communication of data between the various processes. After all internal calculations are performed, the main outputs of CTAS are:

- **Landing Schedules:** CTAS performs runway allocation for all arriving aircraft and produces a time-line schedule and sequence for each runway. Scheduling is initially performed in the Center Area by the TMA and the output is graphically displayed on the TGUI which is used by the TMC. Once in the TRACON area, FAST recomputes and overrides the previous schedule. The new schedule is then displayed on the PGUI.
- **Advisories:** CTAS also computes heading, altitude and speed advisories. DA provides the advisories in Center airspace whereas FAST provides advisories in the terminal area. The advisories are displayed on the GUIs and are then transmitted by voice from the controllers to the aircraft.

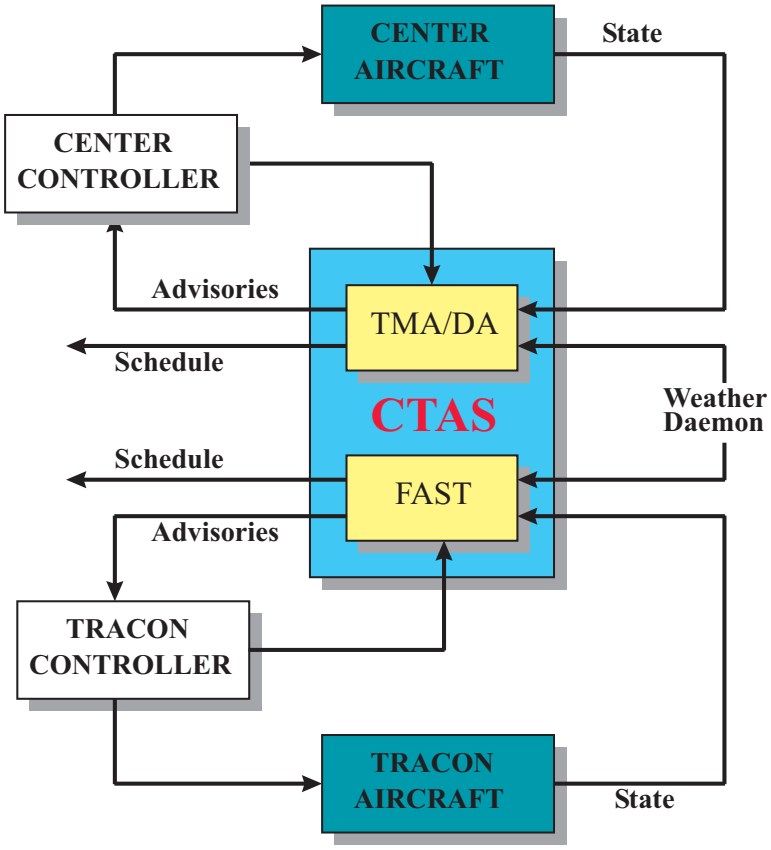


Fig. 1. CTAS Architecture

3 Hybrid Input-Output Automata

In this section we give a brief overview of a model for the CTAS system. In Section 3.1 we outline the modeling formalism (a formal discussion can be found in [8]) and discuss the features that make it ideal for modeling the CTAS system. In Section 3.2 we show how models can be constructed in this framework for one of the CTAS components, the FAST algorithm.

3.1 Overview of the Modeling Framework

Based on the work of [8], we consider a hybrid automaton, A, as a dynamical system that describes the evolution of a finite collection of variables, V_A. Variables are typed; for each $v \in V_A$ let $type(v)$ denote the type of v. For each $Z \subseteq V_A$, a *valuation* of Z is a function that to each $v \in Z$ assigns a value in $type(v)$. Let $\mathbf{Z}$ denote the set of valuations of Z; we refer to $s \in \mathbf{V_A}$ as a system

state. In this paper we assume that the evolution of the variables is over the set $T^{\geq 0} = \{t \in \mathbb{R} | t \geq 0\}$. The evolution of the variables involves both continuous and discrete dynamics. Continuous dynamics are encoded in terms of *trajectories* over V_A, that is functions that map intervals of the time axis to $\mathbf{V_A}$. Discrete dynamics are encoded by *actions*. Upon the occurrence of an action the system state instantaneously "jumps" to a new value. We use Σ_A to denote the set of actions that affect the evolution of A.

Formally, a *hybrid automaton*, $A = (U_A, X_A, Y_A, \Sigma_A^{in}, \Sigma_A^{int}, \Sigma_A^{out}, \Theta_A, \mathcal{D}_A, \mathcal{W}_A)$, is a collection of:

- Three disjoint sets U_A, X_A, and Y_A of variables, called *input*, *internal*, and *output variables*, respectively. We set $V_A = U_A \cup X_A \cup Y_A$.
- Three disjoint sets Σ_A^{in}, Σ_A^{int}, and Σ_A^{out} of actions, called *input*, *internal*, and *output actions*, respectively. We set $\Sigma_A = \Sigma_A^{in} \cup \Sigma_A^{int} \cup \Sigma_A^{out}$.
- A non-empty set $\Theta_A \subseteq \mathbf{V_A}$ of *initial states*.
- A set $\mathcal{D}_A \subseteq \mathbf{V}_A \times \Sigma_A \times \mathbf{V}_A$ of *discrete transitions*.
- A set $\mathcal{W}_A$ of *trajectories* over V_A.

Some technical axioms are imposed on the above sets to guarantee that the definitions are consistent.

An *execution*, α, of the hybrid automaton A is a finite or infinite alternating sequence $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$, where for all i, $a_i \in \Sigma_A$, $w_i \in \mathcal{W}_A$ defined over a left closed time interval, $fstate(w_0) \in \Theta_A$, if α is a finite sequence then it ends with a trajectory and if w_i is not the last trajectory its domain is right-closed and $(lstate(w_i), a_{i+1}, fstate(w_{i+1})) \in \mathcal{D}_A$. Here $fstate(w)$ and $lstate(w)$ denote the initial and final states of a trajectory w. An execution is called *finite* if it is a finite sequence and the domain of its final trajectory is a right-closed interval. A state $s \in \mathbf{V_A}$ is called *reachable* if it is the last state of a finite execution.

To capture the evolution of an HIOA from the point of view of the "outside world" the notion of a *trace* is introduced. Roughly speaking a trace is an execution projected to the external (input and output) variables and actions. Two automata A and B are called *comparable* if they have the same external interface. If A and B are comparable, then we say that A *implements* B if the hybrid traces of A are a subset of those of B. Typically one thinks of B as a *specification* and A as an *implementation* of the specification. A specification is usually a more abstract description that imposes weaker restrictions on the system behavior. Proving that one hybrid automaton implements another can be a complicated task. Usually it is broken up in a series of steps, where the abstract specification is progressively refined. At each step implementation is proved using simulation relations. A *simulation* from A to B is a relation $R \subseteq \mathbf{V}_A \times \mathbf{V}_B$ such that (roughly speaking) two states, r of A and s of B, are related through R if from state s B can reproduce any move that A makes from r (discrete or continuous) by a hybrid execution which is indistinguishable from the move of A from the point of view of the outside world. The final states of the two moves should again be related by R.

Hybrid automata "communicate" through shared variables and shared actions. Consider two automata A and B with $X_A \cap V_B = X_B \cap V_A = Y_B \cap Y_A = \emptyset$

and $\Sigma_B^{int} \cap \Sigma_A = \Sigma_A^{int} \cap \Sigma_B = \Sigma_A^{out} \cap \Sigma_B^{out} = \emptyset$. Under some mild technical assumptions, the *composition*, $A \times B$, of A and B can be defined as a new hybrid automaton with $U_{A\times B} = (U_A \cup U_B) \setminus (Y_A \cup Y_B)$, $X_{A\times B} = X_A \cup X_B$, $Y_{A\times B} = Y_A \cup Y_B$ (similarly for the actions). $\Theta_{A\times B}$, $\mathcal{D}_{A\times B}$ and $\mathcal{W}_{A\times B}$ are such that the executions of $A \times B$ are also executions of each automaton when restricted to its variables and actions. It can be shown that composition respects implementation.

A *derived variable* of A is a function on $\mathbf{V_A}$. Derived variables will be used to simplify the system description, but also to facilitate the analysis. A *property* of A is a boolean derived variable. A property is *stable* if whenever it is true at some state it is also true at all states reachable from that state. A property is *invariant* if it is true at all reachable states. Typically properties will be shown to be stable or invariant by an induction argument on the length of an execution.

In some places differential equations will be used to simplify the description of the set $\mathcal{W}_A$. In such cases $\mathcal{W}_A$ is assumed to be populated by all trajectories generated by the differential equation in the usual way. To simplify the description of $\mathcal{D}_A$, we will assign a *precondition* and an *effect* to each action. The precondition is a predicate on $\mathbf{V_A}$ while the effect is a predicate on $\mathbf{V_A} \times \mathbf{V_A}$. The action can take place only from states that satisfy the precondition; moreover, the states before and after the transition should be such that the effect is satisfied.

The following properties of the HIOA formalism are especially useful for CTAS modeling:

1. **Descriptive Power**: The modeling formalism provides a uniform framework in which one can describe the evolution of general classes of variables, ranging from real and integer numbers (with their associated mathematical structure) to abstract high level data types typically found in a computer program.
2. **Hybrid Dynamics**: The formalism allows us to capture continuous and discrete dynamics and the interaction between the two.
3. **Compositionality**: The modeling formalism allows us to build up the description of the complicated CTAS system by combining simpler entities.
4. **Abstraction**: The notion of specification and implementation allows us to describe the system at various levels of abstraction. This provides a way of showing that the CTAS code satisfies a specification through a sequence of progressive refinement. Abstraction also allows us to structure proofs hierarchically, with simulation relations connecting the levels of the proof hierarchy.

3.2 Formal CTAS Model

The CTAS system will be modeled as an interconnection of a number of components (Figure 2). In this section we will show how the input–output interaction between these components can be captured by appropriate hybrid automata. The models given here will be "high level"; in a number of places we will use

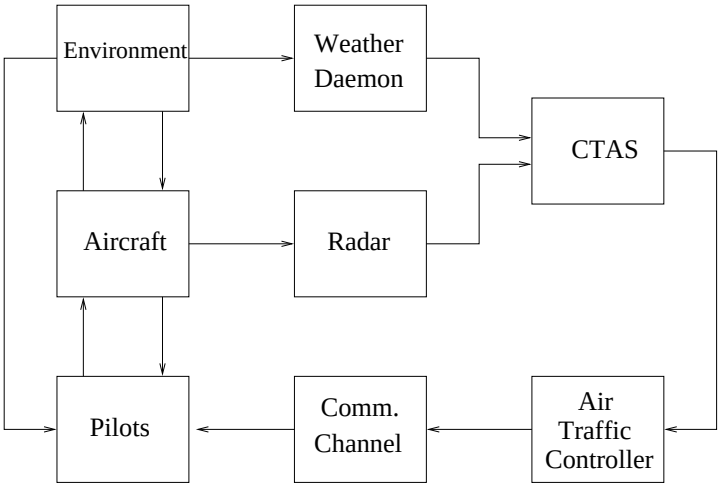


Fig. 2. CTAS Feedback Loop

derived variables to represent parts of the algorithm, the air traffic controller behavior, etc. for which we can not provide explicit expressions at this stage. The model can be refined until it contains sufficient details to carry out meaningful safety analysis. The refinement primarily involves providing explicit expressions for these derived variables. Here we will briefly discuss the operation of all the components shown in Figure 2. For the time being we restrict our attention to FAST; the remaining CTAS components can be similarly modeled. Examples of HIOA pseudo-code for FAST can be found in the appendix; for more examples see [9].

Aircraft Model: The system we consider consists of N aircraft, labeled $1, \ldots, N$. Each aircraft, i, is modeled by a hybrid automaton, $A_i = (U_{A_i}, X_{A_i}, Y_{A_i}, \Sigma^{in}_{A_i}, \Sigma^{int}_{A_i}, \Sigma^{out}_{A_i}, \Theta_{A_i}, \mathcal{D}_{A_i}, \mathcal{W}_{A_i})$. At this stage we assume that the aircraft evolution is not affected by any actions, input, output or internal ($\Sigma_{A_i} = \Sigma^{in}_{A_i} \cup \Sigma^{int}_{A_i} \cup \Sigma^{out}_{A_i} = \emptyset$ and hence $\mathcal{D}_{A_i} = \emptyset$). At a later stage appropriate actions can be added to model discrete changes in the physical system, such as malfunctions.

Each aircraft is identified by its type, for example, TurboJet, 747, etc. This information is stored in an internal variable $Type_i$. The physical movement of the aircraft is summarized by the trajectories of its position and velocity. Let $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $v_i = (v_i^x, v_i^y, v_i^z) \in \mathbb{R}^3$ be the position and velocity of the aircraft with respect to some fixed reference frame on the ground. The motion of the aircraft is influenced by the commands of the pilot and the environmental conditions. Let a_i represent the pilot commands (for the engine, control surfaces, flaps, etc.) and w_i the environmental conditions at the current position of aircraft i (wind and temperature for example). We assume that all trajectories in $\mathcal{W}_{A_i}$

satisfy the differential equation:

$$\begin{bmatrix} \dot{p}_i(t) \\ \dot{v}_i(t) \end{bmatrix} = \begin{bmatrix} v_i(t) \\ f(\mathit{Type}_i, v_i(t), a_i(t), w_i(t)) \end{bmatrix} \quad (1)$$

We set $Y_{A_i} = \{\mathit{Type}_i, p_i, v_i\}$, $U_{A_i} = \{a_i, w_i\}$ and $X_{A_i} = \emptyset$. The function f returns the acceleration of the aircraft, which will in general depend on the aircraft type, the aircraft velocity, the commands of the pilot and the weather conditions.

The aircraft automaton is only partly specified at this stage. We still need to provide an expression for the derived variable f. This expression is likely to be very complicated. For the preliminary safety analysis we start with simple formulas (such as $f = a_i$); more accurate expressions can be obtained from the aircraft model database used by CTAS.

Environment Model: We assume that the motion of each aircraft is influenced by the wind and temperature at its current position. To encode this information we introduce a hybrid automaton $E = (U_E,\ X_E,\ Y_E,\ \Sigma_E^{in},\ \Sigma_E^{int},\ \Sigma_E^{out},\ \Theta_E,\ \mathcal{D}_E,\ \mathcal{W}_E)$. The environment automaton has no internal, input or output actions ($\Sigma_E = \emptyset$) and no internal variables ($X_E = \emptyset$). Its inputs are the positions of all aircraft, ($U_E = \{p_i\}_{i=1}^N$) and its outputs (denoted by $w_i \in \mathbb{R}^4$) are the wind magnitude and direction and the temperature at each one of these positions ($Y_E = \{w_i\}_{i=1}^N$). The environmental conditions are encoded by a function $W : \mathbb{R}^+ \times \mathbb{R}^3 \to \mathbb{R}^4$, that returns the wind and temperature at the current time and the given location.

The environment model is only partly specified at this stage. To complete the description we need to provide an expression for the derived variable W. We propose to start with very simple expressions (e.g. W constant as a function of time) and refine them at a later stage, as the description of the automata that make use of the weather information n (the aircraft and the weather daemon soon to be specified) becomes more detailed.

Radar Automaton: CTAS obtains information about the state of each aircraft through radar. We model the radar by a hybrid automaton $R = (U_R,\ X_R,\ Y_R,\ \Sigma_R^{in},\ \Sigma_R^{int},\ \Sigma_R^{out},\ \Theta_R,\ \mathcal{D}_R,\ \mathcal{W}_R)$. The input variables of R are the positions and velocities of all aircraft ($U_R = \{p_i, v_i\}_{i=1}^N$) while the output variables of R are estimates of these quantities, denoted by $\hat{p}_i$ and $\hat{v}_i$ ($Y_R = \{\hat{p}_i, \hat{v}_i\}_{i=1}^N$). At this stage the radar automaton is assumed to have no input or internal actions ($\Sigma_R^{in} = \Sigma_R^{int} = \emptyset$).

The information that the radar provides about the aircraft is quantized spatially and sampled temporally. We assume that the output variables of the radar automaton fall within an interval centered at the "correct" values dictated by the actual state of the system. Let $n_P \in \mathbb{R}^3$ and $n_V \in \mathbb{R}^3$ denote the width of the intervals for $\hat{p}_i$ and $\hat{v}_i$ respectively. At this stage we assume n_P and n_V are constant; these quantities may become internal variables later on, to model variations of the accuracy of sensing with position and environmental conditions, for example. The output variables of the radar are updated every $\hat{T}_r$ seconds, upon the occurrence of an output action Sample_r. We set $\Sigma_R^{out} = \{\mathit{Sample}_r\}$. An internal variable $T_r \in \mathbb{R}$ keeps track of the time that has elapsed since the

last sample. $\hat{T}_r$ is typically of the order of a few seconds. Once values for n_P, n_V and $\hat{T}_r$ are available the radar automaton will be completely specified.

Weather Daemon: CTAS also obtains information about the environmental conditions in the vicinity of each aircraft. This information is provided by a hybrid automaton $D = (U_D, X_D, Y_D, \Sigma_D^{in}, \Sigma_D^{int}, \Sigma_D^{out}, \Theta_D, \mathcal{D}_D, \mathcal{W}_D)$. D is very similar to the radar automaton R. D has no input or internal actions ($\Sigma_D^{in} = \Sigma_D^{int} = \emptyset$). Its input variables are the weather conditions at the location of each aircraft ($U_R = \{w_i\}_{i=1}^N$) and its output variables are estimates of these quantities denoted by $\hat{w}_i$ ($Y_R = \{\hat{w}_i\}_{i=1}^N$). $\hat{w}_i$ are quantized to within $n_w \in \mathbb{R}^4$ of the real environmental conditions and sampled every $\hat{T}_w$ time units, upon the occurrence of an output action $Sample_w$.

The FAST Automaton: The FAST algorithm itself will be encoded by a hybrid automaton $FAST = (U_{FAST}, X_{FAST}, Y_{FAST}, \Sigma_{FAST}^{in}, \Sigma_{FAST}^{int}, \Sigma_{FAST}^{out}, \Theta_{FAST}, \mathcal{D}_{FAST}, \mathcal{W}_{FAST})$. The input variables of $FAST$ are the output variables of the radar and weather daemon automata and type information from the aircraft automata. Overall, $U_{FAST} = Y_R \cup Y_D \cup \{Type_i\}_{i=1}^N$. FAST provides advisories to the air traffic controllers for all aircraft, $Y_{FAST} = \{adv_i\}_{i=1}^N$. An internal variable, $FAST_AC \subset \{1, \ldots, N\}$, is used to store the labels of all aircraft currently in the TRACON. It is assumed that for aircraft not currently in the TRACON ($i \notin FAST_AC$) a default advisory $adv_i = \perp$ (undefined) is issued. For aircraft in the TRACON, the nature of the advisory depends on the version of FAST. This information is stored in an internal variable $Version \in \{\text{Active}, \text{Passive}\}$. If $Version = \text{Passive}$, the advisory for each aircraft consists of a runway assignment and a landing sequence. If $Version = \text{Active}$, FAST also provides heading, altitude or speed commands. At this stage we model these commands as a position in the $x - y$ plane, P, and a number, V, that encodes a desired heading altitude or speed; the interpretation is that the pilot is asked to guide the aircraft to P and achieve V by the time it gets there. The advisory calculations involve the degrees of freedom available to each aircraft and are restricted by sequencing constraints imposed by the air traffic controllers. This information is stored in internal variables dof_i and $Constraints$. Overall, $X_{FAST} = \{FAST_AC, Constraints, Version\} \cup \{dof_i\}_{i=1}^N$.

The evolution of the FAST automaton is disrupted by input actions. After each action the advisories for all aircraft currently in the TRACON are recalculated. The calculation is encoded by a derived variable $Calculate_Advisory$. Input actions $Sample_r$ and $Sample_w$ are the output actions of the radar and weather daemon automata respectively. Their role is to recalculate the advisories whenever new data becomes available. Input action $Center_Handoff(i)$ occurs when aircraft i enters the TRACON. It represents the hand-off of aircraft from the Center to the TRACON air traffic controllers and is assumed to be the output of a hybrid automaton modeling the air traffic controllers. The effect of $Center_Handoff(i)$ is to add aircraft i to the list of aircraft currently in the TRACON, initialize its possible degrees of freedom and recalculate the advisories for all aircraft. The degrees of freedom are initialized according to a derived variable $Default_dof(i)$, whose "output" will typically depend on the

aircraft type, the point of entry into the TRACON, the weather, etc.

The TRACON air traffic controller influences the evolution of FAST through three input actions. Using action *Constrain_Order*(i, j), the controller can force FAST to schedule aircraft i before aircraft j. The effect of this action is to add (i, j) to the list of sequencing constraints maintained by FAST and recalculate the advisories. *Constrain_dof*(i, dof) allows the controller to reduce the degrees of freedom that FAST considers for aircraft i. Upon occurrence of the action FAST removes the specified degree of freedom from the list dof_i and recalculates the advisories. Finally, the action *Tower_Handoff*(i) occurs when aircraft i lands and is handed-off to the tower controllers. The effect of the action is to remove i from the list of aircraft currently in the TRACON, together with all sequencing constraints involving i on the remaining aircraft. The advisories for the remaining aircraft are recalculated.

To complete the description of the FAST automaton we need to provide expressions for the derived variables *Default_dof*(i) and *Calculate_Advisory*. The expressions need to be extracted from the FAST documentation.

Air Traffic Controller Model: The air traffic controllers are modeled by a hybrid automaton, $ATC = (U_{ATC}, X_{ATC}, Y_{ATC}, \Sigma^{in}_{ATC}, \Sigma^{int}_{ATC}, \Sigma^{out}_{ATC}, \Theta_{ATC}, \mathcal{D}_{ATC}, \mathcal{W}_{ATC})$. The inputs to ATC are the advisories from FAST as well as all the information available for each aircraft. Overall $U_{ATC} = \{adv_i, Type_i, \hat{p}_i, \hat{v}_i, \hat{w}_i\}_{i=1}^{N}$. As at this stage we are only concerned with the FAST operation, ATC will primarily model the TRACON air traffic controllers. The only function of the Center air traffic controllers in this setting is to feed aircraft into the TRACON, by executing action *Center_Handoff*(i). We assume that the Center contains a number of aircraft, whose labels are stored in an internal variable *Center_AC*. An aircraft gets removed from this list and is handed off to the TRACON controller upon the occurrence of output action *Center_Handoff*(i). The precondition of the action is a boolean derived variable *Center_Handoff_Condition*(i).

The TRACON controller may choose not to follow a particular advisory or to follow it after some delay. This information is stored in the boolean internal variables $Follow_i$ and the real internal variables d_i. We assume that the controller keeps track of the previous advisory issued by CTAS for aircraft i in an internal variable $Old_Advisory_i$. $Old_Advisory_i$ is used to trigger an internal action $New_Advisory_i$. Upon occurrence of the action the controller decides whether the new advisory will be followed and selects a delay. If the controller chooses to follow the advisory, the speed, altitude or heading command is transmitted to the pilot of aircraft i after a delay d_i, upon the occurrence of an output action $Send_i$. If the controller chooses not to follow the advisory or if FAST is "passive" the transmitted command is assumed to be determined by a derived variable *Independent_Choice*(i). Our model also allows controllers to issue independent commands in between the FAST advisories, whenever a boolean derived variable *Independent_Choice_Condition*(i) becomes true.

The controller can influence FAST through actions *Constrain_Order*(i, j) and *Constrain_dof*(i, dof). The preconditions for these actions are encoded by boolean

internal variables *Order_Condition*(i, j) and *DOF_Condition*(i, dof). We assume that the controller keeps track of the constraints it has previously issued. This will allow us to make the controller model more realistic later on (for example, require that the controller does not issue contradictory constraints). Finally, the TRACON controller decides when the aircraft has landed and hands it off to the tower controllers. This "action" is encoded by *Tower_Handoff*. The precondition for this action is a boolean derived variable *Tower_Handoff_Condition*(i).

The controller model requires expressions for *Center_Handoff_Condition*(i), *Order_Condition*(i, j), *DOF_Condition*(i, dof), *Independent_Choice_Condition*(i), *Independent_Choice*(i) and *Tower_Handoff_Condition*(i). Obtaining expressions for these variables is likely to be a major challenge, as it involves understanding the complicated decision making process of the human air traffic controllers. To start the safety analysis we will assume that FAST is active, the controller always follows the proposed advisories, never imposes additional constraints and hands off the aircraft to the tower at the runway threshold.

Communication Channel: Communicating commands to the pilots is achieved through communication channel automata, C_i. Each automaton has an input action *Send*$_i$(command), whose effect is to store the command together with a time stamp in an internal multi-set. The message is delivered (and removed from the multi set) upon occurrence of the output action *Receive*$_i$(command). Delivery is guaranteed by at most d_i^c time units from the time the message was sent.

Pilot Model: Finally, the pilot is modeled by a hybrid automaton P_i. P_i accepts input information about the aircraft and the air traffic controller commands (obtained through the input action *Receive*$_i$(command)) and produces input a_i for the aircraft automaton. Similar to the air traffic controllers, a pilot is given the freedom to ignore an ATC command. His/her decision is stored in an internal variable *Follow*$_i^p$. If the pilot chooses to follow a particular command he/she responds after some delay (encoded by input variable d_i^p). In this case, a_i is chosen according to a derived variable *Comply*. Otherwise, a_i is chosen according to a derived variable *Not_Comply*. Expressions for these derived variables are needed to complete the description of the pilot automaton. These expressions may again be hard to obtain as they involve modeling the response of the human pilots and/or the autopilots.

4 Safety Notions

The performance evaluation of large scale systems like CTAS is a very complex process. Various metrics quantitatively measure system performance and allow comparisons between different designs. The three most prominent performance areas for CTAS are:

- **Safety**, which receives top priority
- **Economic considerations**, such as minimizing fuel and operating costs as well as time delays. Other considerations, such as passenger comfort can also be included in this category.

– **Reduction of controller workload** and, more generally, increasing situational awareness of controllers.

Even though all three aspects of the system performance are important, and the interaction between them is very interesting, here we will concentrate on questions of safety. We classify of safety questions into:

– **Nominal Safety:** considers safety under nominal conditions
– **Robust Safety:** questions the robustness of the nominal safety claims,
– **Structural Safety:** questions of safety under structural changes in CTAS
– **Degraded Safety:** considers safety questions in degraded operation.

The above classes of safety measures will be used to determine not whether CTAS is safe but whether CTAS is safer than the current system. The outcome may also depend on the metric used. For example, CTAS may be safer than the current system under nominal operation but not as safe in degraded operation.

For the time being we restrict our attentions to safety questions when the system operation is nominal (in a sense "perfect"). We assume that operation is nominal if:

– **Nominal CTAS**: We start with fixed and reliable version of the CTAS algorithms.
– **Faultless Operation**: There are no hardware malfunctions, no emergency situations (such as aircraft low on fuel), and the environmental conditions are benign.
– **Accurate Models**: The models used by CTAS can accurately predict aircraft movement. This includes the aircraft dynamical models and the weather models. In addition there is no uncertainty in sensors or parameters. For nominal analysis both controllers and pilots can be modeled by a variable delay that nondeterministically takes values in a bounded interval.

Under nominal conditions we can ask the following very precise safety questions which can be thought of as the CTAS nominal safety specification:

– **Completeness:** Will CTAS issue an advisory in every situation?
– **Consistency:** Will CTAS issue the same advisory in identical situations? Consistency is related to controller workload since system predictability increases situational awareness.
– **Stability:** Are the CTAS outputs stable? This is also related to controller workload since advisory changes reduce situational awareness.
– **Separation Requirements:** Loss of separation could be catastrophic and cannot be tolerated.
– **Implementability:** Are the CTAS advisories implementable? Do CTAS advisories satisfy constraints imposed by aircraft dynamics (e.g. stall conditions)?
– **Delay:** What is the effect of delay (in the radar, weather daemon, controller and pilot responses) in the system?

– **Capacity Limits:** What is the maximum possible TRACON capacity or runway acceptance rate for which CTAS can maintain safety? This is related to cost/benefit analysis.

The above list of high level CTAS specifications is refined to the lower levels of the hierarchical structure shown in Figure 3, to derive nominal safety specifications for the CTAS subsystems. Specifications at the level of TMA, DA and FAST can be further decomposed into specifications for lower subsystems and functions (the Route Analyzer (RA), Trajectory Synthesizer (TS), Profile Selector (PFS), Dynamic Planner (DP), etc.) resulting in a set of nominal safety specifications for each component.

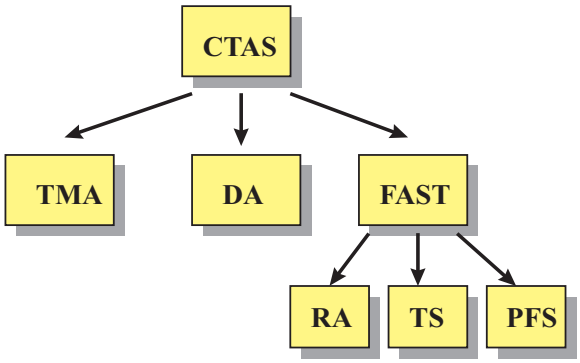


Fig. 3. CTAS Hierarchical Specification Refinement

Whether CTAS satisfies the specifications will depend on the initial configuration and the system parameters. *Our safety analysis methodology for nominal safety will determine the range of configurations and parameter values for which the CTAS advisories are safe.* For example, this will involve determining the rate at which FAST can accept and safely land aircraft that enter through the TRACON gates, for a given runway configuration. As the flow of aircraft to the TRACON gates is determined by the Center TMA, the TMA must in turn guarantee that this flow constraint is not violated. Nominal safety notions try to determine conditions under which the nominal system meets the desired specification. In general the more relaxed the conditions are, the safer the nominal system is. For example, if CTAS can safely handle a flow rate of 60 aircraft an hour in the TRACON under nominal conditions, then it is likely to be more robust than a similar system that can safely handle 50 aircraft an hour.

Given the nominal safety specifications for the various CTAS systems, the next three classes of safety notions try to measure the effect of uncertainty, structural changes and failures to the nominal safety issues.

5 Conclusions

In this paper, a framework for the modeling, specification and safety analysis of the Center-TRACON Automation System (CTAS) is proposed. We believe that this "system theoretic" perspective can prove very fruitful not only for the CTAS problem, but also more generally for the verification of complex, hybrid software systems (see for example [2] for the application of this methodology to the Traffic Alert and Collision Avoidance System (TCAS)). The discussion presented in this paper is only a first step in the verification process of CTAS. Some of the safety questions we formulate are challenging and may require extending the state-of-the-art analysis and verification techniques.

Acknowledgments: The authors would like to thank Darren Cofer and Rosa Weber from Honeywell Technology Center and Nancy Lynch from the Laboratory for Computer Science at MIT for their contributions at various phases of this project.

References

1. H. Erzberger, T.J. Davis, and S. Green. Design of center-TRACON automation system. In *Proceedings of the AGARD Guidance and Control Symposium on Machine Intelligence in Air Traffic Management*, Berlin, Germany, May 1993.
2. John Lygeros and Nancy Lynch. Towards the formal verification of the TCAS conflict resolution algorithms. In *IEEE Control and Decision Conference*, pages 1829–1834, 1997.
3. W. Nedell and H. Erzberger. The traffic management advisor. In *American Control Conference*, San Diego, CA, December 1990.
4. Steven M. Green, Robert A. Vivona, and Beverly Sanford. Descent advisor preliminary field test. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Baltimore, MD, August 1995.
5. T.J. Davis, K.J. Krzeczowski, and C. Bergh. The final approach spacing tool. In *Proceedings of the 13th IFAC Symposium on Automatic Control in Aerospace*, Palo Alto, CA, September 1994.
6. S. Green, T. Goka, and D.H. Williams. Enabling user preferences through data exchange. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, New Orleans, LA, August 1997.
7. Radio Technical Commission for Aeronautics. Final report of RTCA Task Force 3: Free Flight Implementation. Technical report, RTCA, Washington, DC, October 1995.
8. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, number 1066 in LNCS, pages 496–510. Springer Verlag, 1996.
9. George Pappas, John Lygeros, Shankar Sastry, and Nancy Lynch. Modeling, specification and safety analysis of CTAS. Technical Report NEXTOR Research Report RR-97-5, University of California at Berkeley, Berkeley, CA, September 1997.

A FAST Automaton Pseudo-Code

Data Types:

Runways $= \{17L, 17R, \ldots\}$
Types $= \{\text{TurboJet}, 747, DC-10, \ldots\}$
Aircraft $= \{1, \ldots, N\} \subset \mathbb{N}$
Commands $= \{(P, V, K)\}$ with $P \in \mathbb{R}^2$, $V \in \mathbb{R}$, $K \in \{\text{Heading}, \text{Speed}, \text{Altitude}\}$
$\text{Commands}_\perp = \text{Commands} \cup \{\perp\}$
Advisories $= \{(r, s, c)\}$ with $r \in$ Runways, $s \in \{1, \ldots, N\}$, $c \in \text{Commands}_\perp$
$\text{Advisories}_\perp = \text{Advisories} \cup \{\perp\}$
Weather $= \{(\text{Wind}, \text{Temperature})\} \subset \mathbb{R}^4$

Variables:

Input:
$\hat{w}_i \in$ Weather for all $i \in$ Aircraft
$\hat{p}_i \in \mathbb{R}^3$ for all $i \in$ Aircraft
$\hat{v}_i \in \mathbb{R}^3$ for all $i \in$ Aircraft
$Type_i \in$ Types for all $i \in$ Aircraft

Internal:
$FAST_AC \subset$ Aircraft, initially $\emptyset$
$dof_i \subset$ DOF, initially $\emptyset$
$Constraints \subset$ Aircraft $\times$ Aircraft, initially $\emptyset$
$Version \in \{\text{Active}, \text{Pasive}\}$, initially arbitrary

Output:
$adv_i \in \text{Advisories}_\perp$ for all $i \in$ Aircraft, initially $\perp$

Derived:
$Default_dof(i) \subset$ DOF, for all $i \in$ Aircraft
$Calculate_Advisory \in$ **Bool**

Actions:

Input:
e, the environment action
$Center_Handoff(i)$, $i \in$ Aircraft
$Constrain_Order(i, j)$, $i, j \in$ Aircraft
$Constrain_dof(i, \text{dof})$, $i \in$ Aircraft, dof $\in$ DOF
$Tower_Handoff(i)$, $i \in$ Aircraft
$Sample_s$
$Sample_w$

Discrete Transitions:

e:
Effect: arbitrarily reset the input variables
$Center_Handoff(i)$:
Effect:

$FAST_AC := FAST_AC_i \cup \{i\}$
$dof_i := Default_dof(i)$
for $j \in FAST_AC$, choose adv_j so that *Calculate_Advisory* becomes true

Constrain_Order(i, j):

Effect:
$Constraints := Constraints \cup \{(i, j)\}$
for $j \in FAST_AC$, choose adv_j so that *Calculate_Advisory* becomes true

Constrain_dof$(i, \mathbf{dof})$:

Effect:
$dof_i := dof_i \setminus \{\text{dof}\}$
for $j \in FAST_AC$, choose adv_j so that *Calculate_Advisory* becomes true

Tower_Handoff(i):

Effect:
$FAST_AC := FAST_AC \setminus \{i\}$
$dof_i := \emptyset$
$Constraints := Constraints \setminus \cup_{j=1}^{N} (\{(i, j)\} \cup \{(j, i)\})$
for $j \in FAST_AC$, choose adv_j so that *Calculate_Advisory* becomes true

$Sample_s$ and $Sample_w$:

Effect:
for $j \in FAST_AC$, choose adv_j so that *Calculate_Advisory* becomes true

Trajectories:

Input variables follow arbitrary trajectories
Output variables remain constant
Trajectories stop once the precondition of *Tower_Handoff*(i) becomes true

This article was processed using the LaTeX macro package with LLNCS style