Verification of an Autonomous Reliable Wingman Using CCL

Stephen Waydo†
Control and Dynamical Systems
California Institute of Technology
Pasadena, CA 91125, USA
waydo@cds.caltech.edu

Eric Klavins*
Department of Electrical Engineering
University of Washington
Seattle, WA 98195, USA
klavins@ee.washington.edu

***Abstract*— We present a system of two aircraft, one human-piloted and one autonomous, that must coordinate to achieve tasks. The vehicles communicate over two data channels, one high rate link for state data transfer and one low rate link for command messages. We analyze the operation of the system when the high rate link fails and the aircraft must use the low rate link to execute a safe "lost wingman" procedure to increase separation and re-acquire contact. In particular, the protocol is encoded in CCL, the Computation and Control Language, and analyzed using temporal logic. A portion of the verified code is then used to command the unmanned aircraft, while on the human-piloted craft the protocol takes the form of detailed flight procedures. An overview of the implementation for a June, 2004 flight test is also presented.**

I. INTRODUCTION

In modern autonomous systems such as Uninhabited Aerial Vehicles (UAV's), the implementation of the command and control code is an integral part of the control system and as such needs to be analyzed as part of the system. Control algorithms and decision logic are often designed and analyzed using one set of tools (i.e. hybrid systems theory) and then implemented in a language such as C/C++ that may not be well-suited to analysis. At some point in this process a "leap of faith" is required to believe that the real system actually implements the system as analyzed. As these systems become more complex and operate in more uncertain environments it is becoming increasingly desirable to narrow or eliminate the gap between algorithms as analyzed and software as implemented and thus reduce this leap of faith. An appropriate analytical framework is needed to consider these problems, particularly in the context of understanding behavior under failures.

To demonstrate a method for approaching these tasks, we use the Computation and Control Language (CCL) to analyze a realistic example problem, that of the operation of a UAV as a "reliable wingman" in conjunction with a human-piloted craft. We envision a scenario in which a human-piloted lead aircraft and an autonomous following aircraft must operate in concert, flying in formation to achieve some objective. To coordinate their actions the aircraft share two data channels: a high-rate link for the exchange of state information and a low-rate link along which they can pass more limited data. In an analogy to fully human flight, the high-rate link is the visual contact the pilots share and the low-rate link is the radio voice communications channel. We analyze the situation shown in Fig. 1 in which the high-rate link is lost and the aircraft must coordinate using the low-rate link to achieve a safe separation in absense of detailed shared state information.

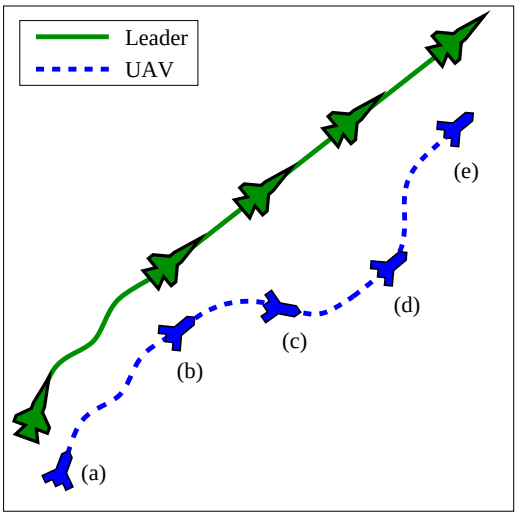

Fig. 1. Lost wingman scenario. At point (a) aircraft are flying in formation. At (b) the data link is lost and the UAV begins executing the lost winman maneuver. At (c) the UAV receives a confirmation message from the leader and turns to match heading. At (d) the data link is restored and at (e) normal operation resumes.

CCL is a specification and implementation language for the operation of concurrent systems that interact both through dynamics and logical protocols. The advantages of this approach are several:

- The formalism of CCL and the analysis techniques allow us to analyze the dynamical behavior and the logical behavior each in an environment naturally suited to them.
- CCL is both a specification and implementation language, meaning that the protocols we analyze and the code we implement are nearly identical; in come cases they are one and the same.
- Reasoning about CCL specifications is done using standard logical tools amenable to the application of automated theorem proving software.
- We can model and reason about portions of the system using CCL specifications even if they will not be implemented as such.

This example will culminate with a flight experiment in

† Support provided by the Fannie and John Hertz Foundation
∗ Support in part by AFOSR grant number F49620-01-1-0361

June, 2004 in which a human-piloted F15 fighter aircraft will lead an autonomous T33 UAV surrogate in a formation flight exercise. The T33 will simulate loss of the state exchange data link and the lost wingman protocol will be tested. Section VI provides an overview of this implementation.

A. Related Work

CCL and the analysis techniques presented here were inspired by UNITY [1] and applications of such ideas to real-time systems. The UNITY formalism is used to specify and reason about concurrent reactive systems [2], but the application of UNITY-like formalisms to dynamic control problems has not been well-developed. Temporal logic has been used to specify and reason about concurrent and distributed systems [3]; with CCL we extend this to include an implementation language.

Provably safe conflict resolution between aircraft has been extensively studied in the context of air-traffic control [4], [5]. The distinction here is that in addition to verifying the safety of the lost wingman maneuver, we reason about the operation of the underlying decision mechanism (the software) to ensure that the assumptions behind the conflict resolution safety proofs are correct.

A great deal of work exists on formation flight [6], [7] and high-performance maneuvering [8] of autonomous aircraft. We are interested in the provably correct and safe implementation of such systems, especially in the presence of faults, requiring an examination of the low-level software in addition to the algorithm specification.

II. CCL: THE COMPUTATION AND CONTROL LANGUAGE

In this paper we use CCL as both a modeling environment for the complete system and as our implementation language for a portion of the software. We summarize here a few important points about CCL; this material has been described in considerably more detail in previous work [9], [10].

A. Structure and Semantics of a CCL Specification

A CCL specification, or *program*, P consists of two parts I and C. I is a predicate on states called the initial condition. C is a set of *guarded commands* of the form $g : r$ where g is a predicate on states and r is a *relation* on states. In a rule, primed variables (such as x'_i) refer to the new state and unprimed variables to the old state. For example,

$$x < 0 : x' > y + 1$$

is a guarded command stating: If x is less than zero, then set the new value of x to be greater than the current value of y plus 1. If x is not less than zero, do nothing.

Programs are composed in a straightforward manner: If $P_1 = (I_1, C_1)$ and $P_2 = (I_2, C_2)$ then $P_1 \circ P_2 = (I_1 \wedge I_2, C_1 \cup C_2)$. This type of composition allows us to modularize our specifications into separate programs such as one for a finite state machine and one to model the dynamics. In an real system, then, we can implement only the state machine and as long as the real dynamics satisfy the CCL specification any proofs about the original specification will remain valid.

In addition to the initial condition and the collection of guarded commands, we need to specify the *semantics* we will use to interpret the specification. The semantics determine how commands are picked for execution, i.e. in what order and with what relative frequency. In general commands are picked in a nondeterministic way to model the uncertainty in how they may be interleaved in the actual system; the semantics we choose place restrictions on this nondeterminism. For the purposes of this paper, where the commands are picked at very close to the same rate, the *EPOCH* semantics will be sufficient. Informally EPOCH semantics can be described as follows:

> **CCL EPOCH Semantics:** Commands are chosen non-deterministically from C, but each command must be chosen once before any command is chosen again.

Thus, the execution of a system is divided into *epochs* during which each command is executed exactly once. This is an attempt to capture the small-time interleaving that may occur between processors that are essentially synchronized, as well as the nondeterministic ordering of commands that can occur when some are picked by some event-driven process while others are picked by a deterministic process. We generally view epochs as occurring at some fixed frequency, although that has not yet been explicitly modelled. In Section III-A we discuss one way to include time in our specification.

The EPOCH semantics are just one of several interpretations of CCL that we are exploring. Other possibilities are that each command executes at approximately the same frequency or each command executes equally many times in any interval; these may be more appropriate when the specification under consideration is executing across multiple processors or agents.

B. Implementation of CCL Specifications

While we may reason about specifications in which the rules are abritrary relations on states, any code we write will be deterministic, meaning that all rules will be assignments. In this case there exists a CCL interpreter known as CCLi [10], [11] that can be used to execute CCL code. The implementation code for all of the programs described here is available online [12].

C. Reasoning about CCL Specifications

We present here a brief overview of the formalism used to reason about CCL specifications, described in more detail in previous work [9]. The definitions of *state*, *state function*, *action*, and *behavior* are taken from [3].

1) States, Variables, and Actions: We begin with a set V of *variable symbols* and a set val of values that the variables may take (natural numbers, real numbers, sets, etc.). A **state** s is a function from V to val, and the set of all states is denoted S. The value of a variable $v \in V$ with respect to a state s is denoted $s[\![v]\!]$. A **state function** f is an expression over symbols in V and constant symbols. The **meaning** of a state function f, denoted $[\![f]\!]$, is a function from states into values and is defined by

$$s[\![f]\!] \triangleq f\,(\forall v \;:\; s[\![v]\!] \;/\; v),$$

that is, the value obtained by replacing all (free occurrences of) variables in f by their values under the state s. A **predicate** is simply a boolean valued state function.

We denote by V' the set $\{v' \;:\; v \in V\}$, that is, the set of all primed variables symbols from V. An **action** is a boolean valued expression over variables in V, primed variables in V' and constant symbols. The **meaning** of an action a, denoted $[\![a]\!]$, is a function from $S \times S$ into values and is defined by

$$s[\![a]\!]t \triangleq a\,(\forall v \;:\; s[\![v]\!] \;/\; v,\; t[\![v]\!] \;/\; v'),$$

that is, the value obtained by replacing all unprimed variables in a by their values under the state s and replacing all primed variables in a by their values under t. Note that we generally regard variables not appearing primed in an action as not changing. A guarded command is simply a particular kind of action. For technical reasons we also allow the action $skip$ in which no values are altered.

We will need to reason about the effect of an action on the set of all states satisfying a predicate. For this, we use the *Hoare triple* notation (standard in Computer Science). If P and Q are predicates and a an action, the Hoare triple relating P to Q by a is defined in CCL as

$$\{P\}a\{Q\} \triangleq \forall s, t \;.\; s[\![P]\!] \wedge s[\![a]\!]t \Rightarrow t[\![Q]\!].$$

2) Behaviors and Temporal Logic: A **behavior** $\sigma : \mathbb{N} \to S$ of a program $P = (I, C)$ is a sequence of states such that $\sigma(0)[\![I]\!]$ and there exists a sequence $\{c_i\}_{i=0}^{\infty}$ of commands in C satisfying the semantics of P such that $\sigma(k)[\![c_k]\!]\sigma(k+1)$.

We reason about entire CCL programs using standard temporal logic [13], [3], which we summarize here. Briefly, *temporal logic formulas* are constructed from predicates, actions, basic connectives (such as $\vee$, $\wedge$, $\neg$ and $\Rightarrow$) and the special operators $\Box$ (always) and $\Diamond$ (eventually). Given a temporal logic formula F, we define $[\![F]\!]$ to be a function from behaviors to $\{true, false\}$ and say that a state s satisfies F if $s[\![F]\!]$. If p is a predicate, a an action, and F a temporal logic formulas, then

1) $\sigma[\![p]\!] \triangleq \sigma(0)[\![p]\!]$,
2) $\sigma[\![\Box F]\!] \triangleq \forall n.\langle \sigma_n, \sigma_{n+1}, ...\rangle[\![F]\!]$.

The formula $\Diamond F$ is equivalent to $\neg\Box\neg F$. We also find the **co** operator, similar to one introduced in [1] and [2], to be useful. If p and q are predicates, then

$$p \;\mathbf{co}\; q \;\triangleq\; \Box\,(p \Rightarrow [(q' \vee skip) \wedge \Diamond q'])$$

Thus, p **co** q (read "p constrains q") means that whenever p is true, then after the next time the state changes, q will be true.

We will generally be interested in when all possible behaviors allowed by a program P and its accompanying semantics M satisfy a temporal logic formula F. If this is the case we write

$$P \models_M F,$$

which we read "P models F under M." If this property is true for the UNITY semantics (which require only that all commands are chosen infinitely often), we simply write $P \models F$.

D. Dynamics and Time

When reasoning about a real-time system it will be necessary to keep track of time and update the dynamics. We can accomplish these tasks simultaneously with a program that updates the state according to a discrete model of the dynamics of the system and increments the time. Consider a general control system

$$\dot{\mathbf{q}} = f(\mathbf{q}, \mathbf{u}), \tag{1}$$

where $\mathbf{q} \in \mathcal{Q}$, $\mathbf{u} \in \mathcal{U}$, and assume that the control $\mathbf{u}$ is held constant over time intervals Δt. Let $\Phi_{\Delta t} : \mathcal{Q} \times \mathcal{U} \to \mathcal{Q}$ be the map of the dynamics through time Δt. To model the flow of time we keep track of two variables, the current time t and the time of the next state update t_n. Each time we update the dynamics using $\Phi_{\Delta t}$ we set the current time to t_n and increment t_n by Δt. As a specification for a CCL program, we have

Program $P_{dyn}(t_0, \Delta t)$			
Initial	$\mathbf{q} \in \mathcal{Q} \wedge \mathbf{u} \in \mathcal{U} \wedge t = t_0 \wedge t_n = t_0 + \Delta t$		
Commands	$true$	:	$\mathbf{q}' = \Phi_{\Delta t}(\mathbf{q}, \mathbf{u})$
			$\wedge t' = t_n$
			$\wedge t'_n = t_n + \Delta t$

Now to model the facts that each command takes some time to execute and there may be time in between the execution of the commands, we append the rule $t' \in (t, t_n)$ to each rule of the system (for space reasons we omit this from the specifications in the remainder of the paper and assume it is present). If we wish to bound the amount of time any command may take by δt, we can *refine* this rule to obtain $t' \in (t, t_n) \cap (t, t + \delta t)$. Assuming no other rules modify t or t_n, we have the following property:

Proposition 1: Let $Q = (I_Q, C_Q)$ be any program such that for all commands $c_q \in C_Q$ and any states s_1 and s_2, $s_1[\![c_q]\!]s_2 \Rightarrow s_1[\![t'_n = t_n \wedge t' \in [t, t_n)]\!]s_2$. That is, a command c_q may not modify t_n and may only modify t by a rule that implies $t' \in [t, t_n)$. Let $P_1 = (I, C) = P_{dyn}(t_0, \Delta t) \circ Q$. Then

$$P_1 \models \Box(t_n - \Delta t \leq t < t_n).$$

Proof: Let $\Pi \equiv (t_n - \Delta t \leq t < t_n)$ be the property in question. Let $c \in C$ be any command and let s_1 and s_2 be any two states such that $s_1[\![c]\!]s_2$. We proceed by cases to show that $\{\Pi\}c\{\Pi\}$:

1) $s_1[\![c]\!]s_2 \Rightarrow s_1[\![t'_n = t_n \wedge t' = t]\!]s_2$:
   In this case $\{\Pi\}c\{\Pi\}$ trivially.
2) $s_1[\![c]\!]s_2 \Rightarrow s_1[\![t'_n = t_n \wedge t' \in [t, t_n)]\!]s_2$:
   First $s_1[\![t'_n = t_n]\!]s_2$, so $s_1[\![\Pi]\!] \Rightarrow s_2[\![t < t_n]\!]$. We have $s_1[\![t' \geq t]\!]s_2$ by assumption and $s_1[\![\Pi]\!] \Rightarrow s_1[\![t \geq t_n - \Delta t]\!]$, so $s_1[\![t' \geq t_n - \Delta t]\!]s_2$. Then also $s_1[\![t' \geq t'_n - \Delta t]\!]s_2$ or simply $s_2[\![t \geq t_n - \Delta t]\!]$. Finally we have $s_2[\![t_n - \Delta t \leq t < t_n]\!]$, or equivalently $s_2[\![\Pi]\!]$ so $\{\Pi\}c\{\Pi\}$.
3) $s_1[\![c]\!]s_2 \Rightarrow s_1[\![t' = t_n \wedge t'_n = t_n + \Delta t]\!]s_2$:
   By simple algebra $s_2[\![t = t_n - \Delta t]\!]$, so $s_2[\![\Pi]\!]$ regardless of s_1 and $\{\Pi\}c\{\Pi\}$.

By assumption on the commands of Q we have covered all possible commands, so for all $c \in C$ we have $\{\Pi\}c\{\Pi\}$. By Lemma 5.2 of [9] then $P \models \Pi$ **co** Π. By the *Initial* section of P_{dyn} we see that $I \Rightarrow \Pi$. Then by Lemma 5.3 of [9] we have $P \models \Box\Pi$, the desired result. ■

III. SYSTEM DESCRIPTION

We now turn to the task of specifying the demonstration system. This system consists of two aircraft, a human-piloted F15 fighter and an autonomous T33 jet trainer serving as a UAV surrogate.

A. Dynamics and Semantics

We will first describe the dynamics of the aircraft in continuous time, then using the technique outlined in Section II-D we will convert this description to a CCL specification and accompanying semantics. We use a simple planar kinematic model to describe each aircraft,

$$\begin{aligned} \dot{x} &= v\cos\psi, \\ \dot{y} &= v\sin\psi, \end{aligned} \tag{2}$$

where $(x, y) \in \mathbb{R}^2$ is the position of the vehicle, $v \in \mathbb{R}^+$ is the speed, and $\psi \in S^1$ is the heading. For the vehicles we analyze, inner-loop controllers regulate the dynamics and we can assume outer-loop actuation of the form

$$\begin{aligned} \dot{\psi} &= u_1, \\ \dot{v} &= u_2. \end{aligned} \tag{3}$$

Thus $\mathbf{q} = (x, y, \psi, v)$ and $\mathcal{Q} = \mathbb{R}^2 \times S^1 \times \mathbb{R}^+$.

We assume that the aircraft share similar performance charactaristics. While this may not be the case in practice (for example, an F15 has dramatically higher capabilities than a T33), it will be necessary to limit the performance of one aircraft to suit the other in order to safely fly in formation. In fact, it may often be most realistic to think of the performance limit as imposed by operating procedures rather than physical limitations. Let the maximum turn rate be $\dot{\psi}_{max}$ and the maximum acceleration be $\dot{v}_{max}$ so that

$$\begin{aligned} \mathcal{U} = \{(u_1, u_2) : -\dot{\psi}_{max} \leq u_1 \leq \dot{\psi}_{max}, \\ -\dot{v}_{max} \leq u_2 \leq \dot{v}_{max}\}. \end{aligned}$$

Using the technique described in Section II-D, we specify two programs, F_{dyn} and T_{dyn}, to keep track of the dynamics of the F15 and T33, respectively, using the subscript 1 to denote the F15 (so for example the position of the F15 is (x_1, y_1)) and 2 to denote the T33. We also constrain the execution to obey the EPOCH semantics, with the additional requirement that each epoch begin with the execution of the command describing the dynamics. In the actual system the execution of each epoch is triggered by an accurate software timer every Δt seconds and each epoch will complete before the next trigger, so this is a realistic model.

B. Controllers

Each aircraft runs a controller that at each update uses its own data plus what is known about the other aircraft to generate controls $\mathbf{u}$. We envision that this is accomplished by some function $control : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathcal{U}$ that we will leave unspecified aside from assumptions about safety properties to be defined later. This allows us to use any number of control techniques in the system to be implemented while retaining the correctness of the safety proofs, provided the implemented controller satisfies the safety specification.

Program P_c	
Initial	$\mathbf{u} = (0, 0)$
Commands	$true : \mathbf{u}' = control(\mathbf{x}_1, \mathbf{x}_2)$

As with the dynamics we denote the F15 controller by F_c and the T33 controller by T_c.

C. Formation Flight

Under normal operation the two aircraft will be flying in formation with one another, with the F15 as a leader and the T33 as a follower. This can be specified by creating a coordinate system with the origin at the F15 and the x-axis oriented with the F15. We then require that the T33 remain within a ball of radius d of some desired tracking point (x_0, y_0) and with an orientation within $\delta\psi$ of that of the F15.

D. Communication

We suppose there are two communications links between the two aircraft, a "high-speed" data connection for state information and a less freqently used "low-speed" connection. The high-speed link is implemented by the lower-level command and control software, and because the CCL program on the T33 interacts with this subsystem only to check if new data have arrived we abstract it into a command c_{data} that updates T_s, the next time data will be sent by the F15, and T, the next time data will be received by the T33. The send time T_s is incremented by

ΔT whenever data are sent, reflecting the fact that the F15 sends data at a regular rate, while the receive time T can be anything in the interval $(T_s, T_s+\tau_d]$, reflecting the uncertain time delay in the system. We keep track of the status of the data link using the boolean variable $data\_on$.

We model the low-speed communications link using a mailbox with a queue and a nondeterministic time delay of up to τ_c. When a message is sent, a record is added to the end of the queue containing a scheduled arrival time $t_a \in (t, t+\tau_c]$. We write $send(i, y)$ as shorthand for

$$\begin{aligned} \tilde{t}'_a &\in (t, t+\tau_c] \\ \wedge queue'_i &= queue_i \# [data = y, t_a = \tilde{t}'_a], \end{aligned}$$

where $\#$ is infix notation for concatenation of an element to the end of a list and i is the index of the mailbox. We then have the predicate $in(i)$ which is $true$ if a message has arrived:

$$in(i) \equiv t \geq (head\ queue_i).t_a.$$

As will be seen below, the nature of the messages we send is such that we are only interested in the most recent message received. We use $msg' = recv(i)$ to denote setting the new value of msg to the $data$ field of the most recent record in $queue_i$ for which $t \geq t_a$ and then deleting from $queue_i$ all messages for which $t \geq t_a$. We use the index 1 for the mailbox of the F15 and 2 for the mailbox of the T33.

All of these communications are specified by the program P_{comm}:

Program P_{comm}

Initial	$T_s = t_0 \wedge T \in (T_s, T_s + \Delta T]$		
Commands	c_{data}	$\equiv$	$t > T \wedge data\_on$:
			$T' \in (T_s + \Delta T, T_s + \Delta T + \tau_d]$
			$\wedge T'_s = T_s + \Delta T$
	$c_{msg,1}$	$\equiv$	$in(1) : msg'_1 = recv(1)$
	$c_{msg,2}$	$\equiv$	$in(2) : msg'_2 = recv(2)$

IV. LOST WINGMAN PROTOCOL

The lost wingman protocol consists of two parts: a CCL state machine managing the T33 and a detailed flight procedure for the pilot of the F15. The flight procedure can also be written as a state machine, and so for purposes of analysis we can model the pilot's behavior using CCL as well.

A. T33

The state machine running on the T33 is depicted in Fig. 2. The system has four modes denoted by the variable m_2 in the programs:

- $normal$, for normal formation flight, abbreviated n
- $lost_1$, for the case where the T33 has ceased receiving state data from the F15 and has not yet received a confirmation of lost status, abbreviated l_1,
- $lost_2$, for the case where lost confirmation has been received from the F15, abbreviated l_2, and

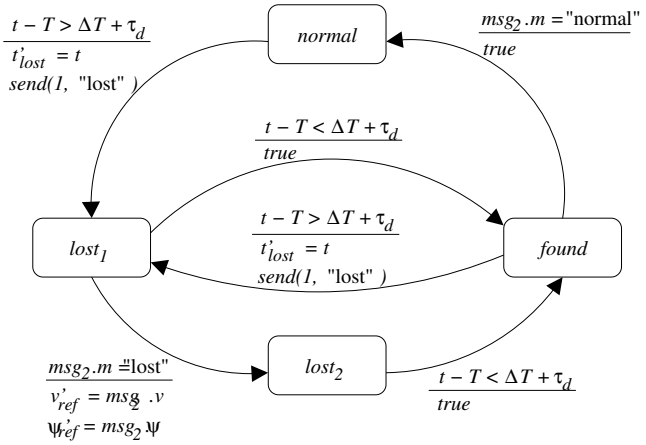

Fig. 2. T33 state machine.

- $found$, for the case where the state data link has been reacquired but normal formation flight has not resumed, abbreviated f.

In this paper we will examine what happens when the T33 enters $lost_1$ mode from $normal$ mode. The portion of the state machine specification involving this portion of operation is:

Program T_{sm}

Initial	$m_2 = n$		
Commands	c_{lost}	$\equiv$	$m_2 \in \{n, f\} \wedge t - T > \Delta T + \tau_d$:
			$m'_2 = l_1 \wedge t'_{lost} = t$
			$\wedge send$(1, "lost")
	c_{found}	$\equiv$	$m_2 \in \{l_1, l_2\} \wedge t - T < \Delta T + \tau_d$:
			$m'_2 = f$
	c_{lost2}	$\equiv$	$m_2 = l_1 \wedge msg_2.m =$ "lost" :
			$m'_2 = l_2 \wedge v'_{ref} = msg_2.v$
			$\wedge \psi'_{ref} = msg_2.\psi$
	...		

1) Normal mode: In $normal$ mode the aircraft are in a formation flight condition. The only transition out of $normal$ mode is to $lost_1$ mode, which occurrs when F15 data has not been received by the latest expected arrival time, i.e. when $t - T > \Delta T + \tau_d$. In this mode we place the formation flight requirement on the T33 controller. Let $(\tilde{x}, \tilde{y}, \tilde{\psi})$ be the position and orientation of the T33 relative to a coordinate system centered at the desired tracking point with the x-axis in the direction of the F15's orientation. We then require

$$\begin{aligned} F_{dyn} \circ F_c \circ T_{dyn} \circ T_c \circ P_{comm} &\models_{EPOCH} \\ \Box(m_2 = n \Rightarrow \|(\tilde{x}, \tilde{y})\|_2 < d \wedge |\tilde{\psi}| < \delta\psi). \end{aligned} \quad (4)$$

We see here how CCL lets us abstract portions of the problem into specifications that can be reasoned about separately. We can use tools from control theory to show that a pair of controllers meet the above specification and then use the results derived below to show safety of the complete system.

2) Lost 1 mode: In $lost_1$ mode, the T33 has ceased receiving state information from the F15. In this mode it commands the controller to execute a pre-specified open-loop escape maneuver. Transitions out of this mode are to the $found$ mode, if state data is reacquired, or to $lost_2$ mode, if an acknowledgement message is received from the F15.

3) Lost 2 mode: In $lost_2$ mode the T33 has received a messsage from the F15 that acknowledges the lost wingman status and includes a reference speed v_{ref} and heading ψ_{ref}. The software then commands the controller to track these references. The transition out of this mode is to $found$ mode when state data is reacquired.

4) Found mode: In $found$ mode the T33 has reaquired the state data link and will command the controller to maintain a safe distance from the F15. The T33 will then periodically send a message to the F15 indicating the found status and thereby requesting to rejoin the formation. The transitions out of this state are back to $lost_1$ mode if data is once again lost or to $normal$ mode if the F15 approves the rejoin request.

B. F15

The F15 state machine consists of just two modes (denoted by m_1 in the specifications), $normal$, for normal formation flight, and $lost$, for the lost wingman scenario. In $normal$ mode the pilot is free to fly at will within a performance envelope that ensures safe formation flight is possible. If the pilot receives a lost message from the T33, the procedure is to transition to straight and level flight and transmit to the T33 the resulting speed and heading. Upon receiving and acknowledging a rejoin request and observing that the T33 has rejoined formation safely, the pilot can resume $normal$ operation.

V. VERIFICATION

We constrain the CCL software on the T33 to satisfy the EPOCH semantics, and further require that each epoch begin with the execution of the command updating the dynamics and time. This is a model of the real system in which the execution of an epoch is triggered by a software timer. We then have a simple result stating that for a program P_1 as defined in Proposition 1 the maximum increment in time between subsequent executions of a given command is less than $2\Delta t$:

Proposition 2: Let σ be any behavior of P_1 and let $\{k_i\}_{i=1}^{\infty}$ be the sequence of steps at which a command c is chosen for execution (i.e. $\sigma^{-1}(c)$ taken as an ordered list). Then

$$\sigma(k_i)[\![t' - t < 2\Delta t]\!]\sigma(k_{i+1}).$$

Proof: By Proposition 1 we know that $\sigma(k_i)[\![t_n - \Delta t \leq t < t_n]\!]$. By the constraint that each epoch begin with the state update command we know that $\sigma(k_i)[\![t_n \leq t' < t_n + \Delta t]\!]\sigma(k_{i+1})$. Together this gives us $\sigma(k_1)[\![t_n - \Delta t \leq t < t_n \wedge t_n \leq t' < t_n + \Delta t]\!]\sigma(k_{i+1})$, which implies $\sigma(k_i)[\![t' - t < 2\Delta t]\!]\sigma(k_{i+1})$. ■

A simple corollary is that for any time t_1 and clause c there exists a k such that $\sigma(k)[\![t \in (t_1 - 2\Delta t, t_1) \wedge c]\!]\sigma(k+1)$.

A. Lost Wingman Scenario

We examine here a bound on the amount of time since receipt of the last state data packet it takes for the T33 to recognize that it is lost and enter lost mode.

Proposition 3: Let $P_2 = T_{dyn} \circ T_{sm} \circ P_{comm}$ and let σ be a behavior of P_2. Suppose there exists a k_1 such that $\sigma(k_1)[\![t - T > \Delta T + \tau_d + 2\Delta t]\!]$, i.e. the time is more than $2\Delta t$ greater than the latest possible packet arrival time. Then $\sigma(k_1)[\![m_2 \in \{l_1, l_2\}]\!]$. Formally,

$$P_2 \models_{EPOCH} \Box(t - T > \Delta T + \tau_d + 2\Delta t \Rightarrow m_2 \in \{l_1, l_2\}).$$

Proof: We examine the command $c_{lost} = g_{lost} : r_{lost}$ from the program T_{sm}. By Proposition 2 there exists a $k_2 < k_1$ such that $\sigma(k_2)[\![t > t' - 2\Delta t]\!]\sigma(k_1)$ and $\sigma(k_2)[\![c_{lost}]\!]\sigma(k_2+1)$, that is the clause c_{lost} was chosen for execution when time time was larger than $\sigma(k_1)[\![t]\!] - 2\Delta t$. Thus $\sigma(k_2)[\![t - T > \Delta T + \tau_d]\!]$, and we examine the two possible cases for m_2 when c_{lost} was chosen:

1) If $\sigma(k_2)[\![m_2 \in \{n, f\}]\!]$ then $\sigma(k_2)[\![g_{lost}]\!]$ and so $\sigma(k_2 + 1)[\![m_2 = l_1]\!]$.
2) If $\sigma(k_2)[\![m_2 \in \{l_1, l_2\}]\!]$ then $\sigma(k_2)[\![\neg g_{lost}]\!]$ and so $\sigma(k_2)[\![m_2' = m_2]\!]\sigma(k_2 + 1)$.

Thus we see that $\{t - T > \Delta T + \tau_d\}c_{check}\{m_2 \in \{l_1, l_2\}\}$. Now the only command transitioning out of $\{l_1, l_2\}$ is c_{found} which has as part of its guard the predicate $t - T < \Delta T + \tau_d$, so for all commands $c \in C$, $\{t - T > \Delta T + \tau_d \wedge m_2 \in \{l_1, l_2\}\}c\{m_2 \in \{l_1, l_2\}\}$. Now for all $k_3 \in [k_2 + 1, k_1]$ we have $\sigma(k_3)[\![t - T > \Delta T + \tau_d \wedge m_2 \in \{l_1, l_2\}]\!]$ and so the result holds. ■

It will be impossible to prove that the two aircraft can never collide if the F15 is never made aware that the T33 is in a lost state. Instead we will need to reason about how soon the F15 receives the "lost" message and responds to it and ensure that the aircraft cannot collide within that time. Let the maximum time required for the F15 to roll level and send a reply (thereby entering $lost$ mode) after receiving a "lost" message be τ_r. The following proposition then gives us a bound on the amount of time between the T33 entering $lost_1$ mode and the F15 entering $lost$ mode:

Proposition 4: Let k_1 be such that $\sigma(k_1)$ is the state immediately after the T33 transitions to l_1 mode, so $\sigma(k_1)[\![t > t_{lost}]\!]$ and $\sigma(k_1)[\![m_2 = l_1]\!]$. Suppose there exists $k_2 > k_1$ such that $\sigma(k_2)[\![t > t_{lost} + \tau_c + \tau_r]\!]$ and for all $k \in [k_1, k_2]$ we have $\sigma_k[\![m_2 \in \{l_1, l_2\}]\!]$. Then $\sigma(k_2)[\![m_1 = l]\!]$.

Proof: We again examine c_{lost}, and see that the T33 sends a "lost" message when it transitions to l_1 mode. The F15 receives this message no more than τ_c seconds later, and itself transitions to l mode no more than τ_r seconds

after that by assumption. Thus at some point in the interval $(t_{lost}, t_{lost} + \tau_c + \tau_r)$ the F15 enters $lost$ mode, or

$$\exists k_3 \,.\, \sigma(k_3)[\![t \in (t_{lost}, t_{lost} + \tau_c + \tau_r) \wedge\ m_1 = lost]\!].$$

Because time is increasing, i.e. $t = t_1$ **co** $t > t_1$, we see that $k_3 \in [k_1, k_2]$.

By assumption $\sigma(k)[\![m_2 \in \{l_1, l_2\}]\!]$ for all $k \in [k_1, k_2]$, and so in particular this is true for k_3. The F15 only transitions out of $lost$ mode when it receives a "found" message from the T33, which is only sent when the T33 transitions to $found$ mode, so

$$(m_2 \in \{lost_1, lost_2\} \wedge m_2 = lost)\ \mathbf{co}\ m_2 = lost.$$

Thus

$$\forall k \in [k_3, k_2] \,.\, \sigma(k)[\![m_2 \in \{lost_1, lost_2\} \wedge m_1 = lost]\!]$$

and the result holds. ■

Proposition 5: Let t_m be the time when the T33 receives a "lost" message from the F15 and t_{l_2} be the time when the T33 transitions to l_2 mode. Then $t_{l_2} - t_m < 2\Delta t$.

Proof: The proof is immediate from the application of Proposition 2. ■

B. Lost Wingman Dynamics

In this section we use the underlying continuous-time dynamics of the aircraft to determine bounds on the positions of the aircraft as they evolve in discrete-time. Let $\Gamma(x, y, \psi, \varphi)$ be the cone with vertex (x, y) oriented in the direction ψ and with half-angle φ.

Proposition 6: Consider an aircraft with dynamics given by P_{dyn}, where $\Phi_{\Delta t}$ is the map of (2) and (3) through time Δt, and let σ be a behavior of P_{dyn}. Then for any $\tau > 0$ and k_1, k_2,

$$\sigma(k_1)\left[\!\!\left[t' = t + \tau \Rightarrow (x', y') \in \Gamma\left(x, y, \psi, \frac{\dot{\psi}_{max}\tau}{2}\right)\right]\!\!\right]\sigma(k_2)$$

Proof: Because P_{dyn} is equivalent to a discrete-time sampling of (2) and (3) it is sufficient to show that

$$(x, y)(t + \tau) \in \Gamma\left(x(t), y(t), \psi(t), \frac{\dot{\psi}_{max}\tau}{2}\right).$$

The geometry of the system for a constant turn rate $\dot{\psi}$ is shown in Fig. 3. After time τ the angle $\Delta\psi = \psi - \psi_0$ is equal to $\dot{\psi}\tau$. The two dotted lines in the figure are radii of the same arc, so their lengths are equal. The angle γ is then $\frac{\pi - \Delta\psi}{2}$, so the angle $\varphi = \frac{\pi}{2} - \gamma = \frac{\Delta\psi}{2} = \frac{\dot{\psi}\tau}{2}$. This angle will be maximized for $\dot{\psi} = \dot{\psi}_{max}$. Thus the aircraft must remain inside the given cone. ■

Now we examine a sufficient condition that guarantees the aircraft cannot collide.

Proposition 7: Suppose that at time t_0 the F15 is located at $(0, 0)$ with orientation in the $+y$ direction and the T33 is within a ball of radius D of the point (x_0, y_0), where $|x_0| > D$ (so the T33 can only be on one side of the F15), with orientation equal to that of the F15 and at this time

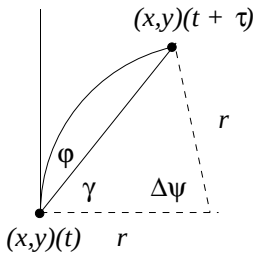

Fig. 3. Geometry of Proposition 6. The aircraft starts at the lower-left corner with orientation straight up and turns at a constant rate $\dot{\psi}$.

the T33 begins executing the lost wingman maneuver (set $|\dot{\psi}| = \dot{\psi}_{max}$, with direction away from the F15). Then if

$$y_0 \sin\frac{\dot{\psi}_{max}\tau}{2} - x_0 \cos\frac{\dot{\psi}_{max}\tau}{2} + D < 0 \qquad (5)$$

is satisfied for all $0 \le \tau \le t - t_0$, there can be no collision between the aircraft before time t.

Proof: Assume that $x_0 > 0$; the rest of the result follows from symmetry. We show that the path of the T33 must lie behind the cone defined by φ determined in Proposition 6. The line defined by φ can be written as $y = \frac{x}{\tan\varphi}$, with points behind the line then given by $y < \frac{x}{\tan\varphi}$. The position of the T33 when $t - t_0 = \tau$ is $(\tilde{x} + \frac{v}{\dot{\psi}_{max}}(1 - \cos\dot{\psi}_{max}\tau), \tilde{y} + \frac{v}{\dot{\psi}_{max}}\sin\dot{\psi}_{max}\tau)$, where $(\tilde{x}, \tilde{y})$ is the actual starting position of the T33. This point is behind the cone boundary if

$$\tilde{y} + r\sin\dot{\psi}_{max}\tau < \frac{\tilde{x} + r(1 - \cos\dot{\psi}_{max}\tau)}{\tan\varphi}.$$

Thus if this inequality is satisfied for all $0 \le \tau \le t - t_0$, the T33 is always outside the region reachable by the F15 by time t_0. The worst case is given by $(\tilde{x}, \tilde{y}) = (x_0 - D\cos\varphi, y_0 + D\sin\varphi)$. Substituting this into the above inequality, noting from Proposition 6 that $\varphi = \frac{\dot{\psi}_{max}\tau}{2}$, and applying trigonometric identities yields the result. ■

We now extend this result to the case where the T33's initial orientation may differ from that of the F15.

Proposition 8: Suppose that at time t_0 the T33 is within a ball of radius d of the point (x_0, y_0), where $|x_0| > d$, with orientation within $\delta\psi$ of the F15 and at this time the T33 begins executing the appropriate lost wingman maneuver. Then if

$$y_0 \sin\frac{\dot{\psi}_{max}}{2} - x_0 \cos\frac{\dot{\psi}_{max}}{2} + d + \delta\psi\frac{v}{\dot{\psi}_{max}}\sqrt{2 - 2\cos\dot{\psi}_{max}\tau} < 0 \qquad (6)$$

is satisfied for all $0 \le \tau \le t - t_0$, there can be no collision between the aircraft before time t.

Proof: We make the same assumptions as in Proposition 7. Let the actual position of the T33 at time t_0 be $(\tilde{x}, \tilde{y})$. If the orientation matches that of the F15, the position at time $t_0 + \tau$ is again $(\tilde{x} + r(1 - \cos\dot{\psi}_{max}\tau), \tilde{y} + r\sin\dot{\psi}_{max}\tau)$. The distance between this point and $(\tilde{x}, \tilde{y})$

is $\frac{v}{\dot{\psi}_{max}}\sqrt{2-2\cos\dot{\psi}_{max}\tau}$. If the initial orientation is perturbed by $\delta\psi$, the resulting perturbation in the final position is then $d_f = \delta\psi\frac{v}{\dot{\psi}_{max}}\sqrt{2-2\cos\dot{\psi}_{max}\tau}$. Thus we can bound the error caused by perturbing the initial orientation by considering it instead as a perturbation of d_f in the initial condition. The total effective perturbation in initial condition is then $D = d + d_f$. Substituting this into (5) yields the result. ∎

For the experimental system, $v = 150m/s$ and $\dot{\psi}_{max} = 0.1rad/s$. If the T33 is attempting to track a point $150m$ away from the F15 and $\frac{\pi}{6}$ radians behind it, Proposition 8 says that if the T33 begins executing the lost wingman procedure while within a $5m$ radius of the desired operating point and within 5 degrees of alignment, collision-free operation is guaranteed for $11s$.

We are now ready to attack the main safety result:

Theorem 1: Suppose data is generated at the F15 every ΔT seconds and received at the T33 with maximum latency τ_d. Suppose the controller is such that under normal operation, if a final packet arrives at $t = T$, then at $t = T + \Delta T + \tau_d + 2\Delta t$ the T33 is within a ball of radius d around the actual desired tracking position and within $\delta\psi$ of the F15's orientation. Then if the condition in (6) holds for d and $\delta\psi$ for $0 \leq \tau \leq 2\Delta t + 2\tau_c + \tau_r$, the T33 will safely pass through $lost$ mode in the event of a packet loss.

Proof: If a packet arrives at time T, the next packet is expected by time $T + \Delta T + \tau_d$. If this packet does not arrive, the T33 will enter $lost$ mode within $2\Delta t$ seconds by Proposition 3. Thus by assumption on the controller, the T33 enters $lost$ mode while within a ball of radius d of the desired tracking point and $\delta\psi$ of the orientation of the F15. By Proposition 4, the F15 will send a "lost" message and transition to $lost$ mode no more than $\tau_c + \tau_r$ seconds later. This message will arrive at the T33 after no more than τ_c seconds, and by Proposition 5 the T33 will enter $lost_2$ mode no more than $2\Delta t$ seconds later. The total time between the T33 entering $lost$ mode and entering $lost_2$ mode is then bounded by $2\Delta t + 2\tau_c + \tau_r$. By Proposition 8 the T33 and the F15 cannot collide within this time. The other possible transition out of $lost_1$ mode is to $found$ mode if new data is received; if this transition occurrs within this time then the result holds trivially. ∎

VI. EXPERIMENTAL IMPLEMENTATION

The software described here will be implemented on a F15 and T33 testbed for a flight test in June, 2004. Low-level control aboard the T33 will be carried out within the framework of Boeing's Open Control Platform (OCP). The OCP will also incorporate a module that executes a portion of the verified CCL code at scheduled intervals. The program T_{sm} and a portion of P_{comm} will be implemented in CCL, while the other programs discussed here serve as specifications for the control code currently under development. The CCL specifications for the F15 will provide the basis for the detailed flight procedures given to the pilot.

VII. CONCLUSIONS AND FUTURE WORK

The results given by Theorem 1 rely on very conservative assumptions about when a collision may occur. The problem of minimal time for the F15 and the T33 to have a possibility of collision is treated more precisely by the two-car problem of differential game theory [14], and so one avenue of future work is to apply results as in [4] to reduce this conservatism.

A major focus of future work will be developing automated tools for the design and analysis of CCL specifications. Opportunities include the development of a graphical design environment for state machines with automated CCL code generation and the use of automated theorem proving assistants such as Isabelle [15] when reasoning about specifications. Such tools will become critically important as the systems we analyze become more complex.

VIII. ACKNOWLEDGMENTS

This work was supported in part by DARPA under the Software Enabled Control program, John Bay program manager, and the Fannie and John Hertz Foundation. We thank Brian Mendel and Jim Paunicka at Boeing for developing the flight test platform and helping us transition this work to the OCP framework. We also thank Richard Murray, John Hauser, Jason Hickey, and Mani Chandy for influencing the ideas described in this work.

REFERENCES

[1] K. Chandy and J. Misra, *Parallel Program Design: A Foundation*. Reading, MA: Addison-Wesley, 1988.

[2] J. Misra, "A logic for concurrent programming: Safety and Progress," *Journal of Computing and Software Engineering*, vol. 3, no. 2, pp. 239–300, 1995.

[3] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[4] C. Tomlin, I. Mitchell, and R. Ghosh, "Safety verification of conflict resolution maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 2, no. 2, pp. 110–120, 2001.

[5] Z.-H. Mau, E. Feron, and K. Billimoria, "Stability of intersecting aircraft flows under decentralized conflict avoidance rules," in *AIAA Conference on Guidance, Navigation, and Control*, 2000.

[6] W. B. Dunbar and R. M. Murray, "Model predictive control of multi-vehicle formations," in *41st IEEE Conference on Decision and Control*, Las Vegas, NV, December 2002.

[7] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," in *2002 IFAC World Congress*, 2002.

[8] J. Hauser and R. Hindman, "Aggressive flight maneuvers," in *36th IEEE Conference on Decision and Control*, San Diego, CA, December 1997.

[9] E. Klavins, "A formal model of a multi-robot control and communication task," in *42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003.

[10] ——, "A language for modeling and programming cooperative control systems," in *Proceedings of the International Conference on Robotics and Automation*, 2004, to Appear.

[11] "The Computation and Control Language (CCL)." [Online]. Available: http://sveiks.ee.washington.edu/ccl/

[12] "Verification of an autonomous reliable wingman using CCL." [Online]. Available: http://www.cds.caltech.edu/~waydo/lost_wingman/

[13] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[14] R. Isaacs, *Differential Games*. Mineola, NY: Dover, 1965.

[15] *Isabelle: A generic theorem prover*, ser. LNCS 828. Springer-Verlag, 1994.