Hybrid Model Visualization in Requirements and Design: A Preliminary Investigation

Jeff Magee, Jeff Kramer, Bashar Nuseibeh
Department of Computing, Imperial College, London SW7 2BZ, UK.
{jnm, jk, ban}@doc.ic.ac.uk

David Bush, Julia Sonander
National Air Traffic Services Ltd, 1 Kemble Street, London WC2B 4AP, UK
{David.Bush, Julia.Sonander}@nats.co.uk

Abstract

This paper reports on a preliminary investigation into applying work on graphic animation of behavioral models to an air traffic control case study – the National Air Traffic Services (NATS) Short Term Conflict Alert (STCA) system that advises controllers of potential conflicts between aircraft in controlled airspace. Graphic animation permits a model to be visualized in the context of a problem domain. The paper describes how, in order to construct a satisfactory visualization, the finite state behavioral model of the STCA system is extended with boolean functions over continuous variables to form a hybrid model. The rationale for constructing this hybrid model and its potential use in requirements and design are discussed.

Keywords
Labeled Transition System, Hybrid Models, Graphic Animation, Air Traffic Control

1. Introduction

A model-based approach to software development involves building analysis models early in the software lifecycle. These models can be developed shortly after initial requirements capture and refined in parallel with further requirements elicitation so that early feedback on the operation of a proposed system can be sought from users and so that potential design problems are highlighted early.

A previous paper[1] proposed graphic animation as a means to communicate model behavior and the results of analysis to non-technical stakeholders. The paper presented a sound semantic basis for associating graphic animations with behavioral models specified using Labeled Transition Systems. The paper also described a flexible toolkit for animating models developed using the Label Transition System Analyzer (LTSA) [2]. In this paper, we describe some initial work on applying the ideas and tools described in [1] to a real case study.

The case study is concerned with a system used in air traffic control. This National Air Traffic Services (NATS) Short Term Conflict Alert (STCA) system, warns air traffic controllers of potential conflicts of the required separation standards for aircraft in controlled airspace. The system already exists and has been deployed in UK air traffic control centers as well as being implemented in other international ATC centers. An existing system was chosen so that we could assess the utility of the model and its associated visualization. In essence, we can evaluate the model against the existing implementation – the reverse of the normal use of models. In addition, as we describe in the following, the system is subject to ongoing development and a successful model will be of use in evaluating changes in advance of implementation.

2. Short Term Conflict Alert (STCA)

The STCA system is safety net system. Normal ATC procedures maintain vertical and lateral separation between aircraft. STCA attempts to predict potential conflicts should this separation not be maintained. The objective is to give the controller sufficient advance warning of a conflict so that he/she can take remedial action. The challenge for the STCA designers is achieving this while minimizing false warnings as controllers would ignore a system that generated too many false alerts.

The system takes as its input information on the tracks of aircraft within a particular region. This track information is provided by the radar data processing units of the air traffic control center and may be computed from the input from more than one radar sensor. The track information for an aircraft consists of its identification code, its position (X-coordinate, Y-coordinate, altitude) and the rate of change of its position, i.e. velocity over the ground and climb/descent rate.

The STCA outputs to an air traffic controller's console by highlighting the pair of aircraft for which a conflict alert has been detected and by displaying a conflict alert indication. The overall architecture of the STCA system is depicted in Figure 1.

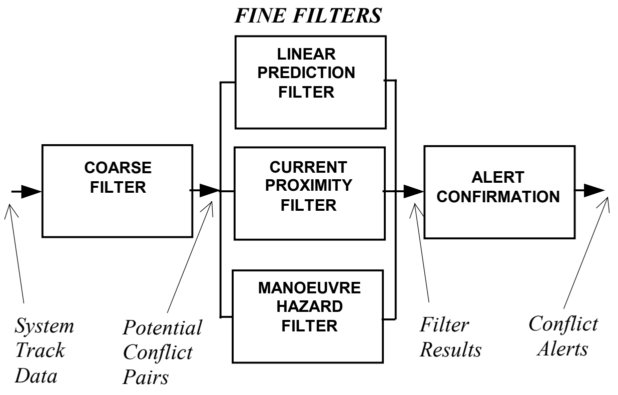

Figure 1 – STCA Architecture

This architecture is designed to reduce the processing load of the STCA system. Track data is initially presented to the *coarse filter*, which examines each pair of tracks and applies a set of criteria as to whether the pair has the potential for future conflict. The filter computes the current separation in distance and altitude between the aircraft that form the pair and in addition, using the velocity/climb information in the track data does a simple linear extrapolation to see if they will potentially conflict in the near future. The look ahead time is a parameter of the system and is typically in the order of two minutes. A pair that fails the coarse filter is not examined further and thus incurs no further processing overhead. The STCA system runs periodically and the full set of tracks is presented to the coarse filter each cycle. Typically the cycle time is 4 or 5 seconds.

Track pairs that pass the coarse filter are submitted to a set of fine filters. These determine with more accuracy and to finer lateral and vertical separation limits, whether the predicted future positions of the aircraft will simultaneously violate lateral and vertical separation. The specific separation distances and heights depend on the region of airspace that the aircraft currently occupy. The *linear prediction* filter assumes that aircraft are proceeding in straight lines along their current track vectors. The *current proximity* filter considers the current separation of the track pair, but also includes a form of prediction that is less sensitive to track velocity than the linear prediction filter. The *manoeuvre hazard* filter detects potentially hazardous manoeuvres between a pair of aircraft in close vertical proximity. The track data contains an indication of whether the aircraft is turning and in which direction.

A track pair that passes any of the filters is a potential conflict. However, further filtering is required to avoid large numbers of transient alerts. This filtering is provided by the *alert confirmation* stage that requires that fine filter "hits" persist over a number of cycles. This number depends on both the filter that detected the hit and the region of airspace occupied. It is a system parameter.

This is necessarily a grossly simplified explanation of what is in reality a complex system. The complexity arises from the requirement to predict the future position of aircraft that are controlled directly by a pilot rather than the air traffic control system. The further the look ahead period, the greater will be the number of false alerts. The system attempts to reduce the number of false alerts by detecting situations that may cause the fine filters to detect hits but which are in reality safe, for example, if aircraft are proceeding in level flight on recognised flight levels or if they are turning away from each other. This sort of condition is used to inhibit or delay alerts in the Alert Confirmation stage.

The parameters and algorithms used in the STCA system are under continuous review by National Air Traffic Services Ltd (NATS) to improve the accuracy of the system. Radar tracks are recorded at each air traffic control center so that any situations that lead to conflict alerts can be analyzed to see if the alert signaled a potential danger. An offline emulator of the STCA system facilitates this analysis. The emulator program is used to explain the reason for the alert and its output can be discussed with controllers to assess the significance of the alert. This output is currently a static document that plots position and height against time for each of the pair of aircraft involved in the alert. The requirement to capture four dimensions on a two dimensional sheet of paper leads to a document that is not always easily understood. It is in this context of analysis that we foresee an immediate use for the STCA model and its associated dynamic visualization that we describe in the next section.

3. STCA Model

Our approach to modeling the STCA system is firstly, to develop a discrete finite state model of the behavior of the system as a Labeled Transition System and then to augment this model with graphic animation that captures more of the continuous behavior of the system. This hybrid model provides the benefits of both support for behavior analysis using LTSA and support for visualization using graphic animation.

3.1 Discrete Model

The discrete model of the system captures its behavior as a set of processes that communicate through shared events. The overall behavior can be formed by sequential and parallel composition of these processes. The difficulty in constructing this sort of model is choosing an appropriate level of abstraction. If we make the model too detailed then state space explosion makes the model intractable for analysis. If the model is too abstract, then the model provides only trivial insights into system operation.

Our major insight in constructing this model was to realize that it is only necessary to consider the behavior of STCA with respect to a single pair of aircraft tracks. The system performs an identical function for each pair of tracks. Consequently, we identify the significant events that relate to a single pair of aircraft and construct the model using this alphabet of events. The significant events we consider include coarse filter pass or fail and fine filter hit or miss events. We abstract from the continuous variables concerned with aircraft position, height and velocity, and model only the events output by the filters as a result of using these variables as input. The processes involved follow naturally from the STCA architecture of Figure 1.

Figure 2 lists a fragment of the discrete model that describes the main processing cycle of STCA. It is specified in FSP [3], a simple process algebra used as the input notation for the Labeled Transition System Analyzer.

```
||FINE_FILTERS = (PROXIMITY
                 || LINEAR
                 || MANOEUVRE
                 || CONFIRM_ALERT
                 ).

/* main processing cycle */

CYCLE = (cycle -> coarse[pass:Bool] ->
         if pass then
            FINE_FILTERS ; CYCLE_END
         else
            (alert[False] -> CYCLE_END)
         ),
CYCLE_END = (end -> CYCLE).
```

Figure 2 – Fragment of STCA model

In the FSP notation of Figure 2, '||' is parallel composition, ';' sequential composition and '->' means action prefix. Action or event identifiers begin with a lower case letter and processes with an uppercase letter. The action `coarse[pass:Bool]` followed by the conditional represents a choice between the two actions `coarse[True]` and `coarse[False]` in the LTS compiled from `CYCLE` as shown in Figure 3.

As mentioned above, we do not model the Coarse Filter directly since this is an algorithm involving continuous variables. Instead we abstract from it by leaving the choice between `coarse[True]` and `coarse[False]` to the environment.

We can use this model that abstracts from the operation of filters to check for correct operation of the logic of the system. Model checking can be used to verify absence of deadlock and that safety properties are preserved – for example, that an alert is not confirmed if a fine filter has not previously detected a conflict hit. However, perhaps the greatest use is in exploring the model interactively to examine the effect of sequences of filter hits and inhibiting conditions. The model, which does not capture the complete discrete behaviour of the system, currently consists of five pages of FSP and comments. It generates an LTS with 25776 state and 89,368 transitions.

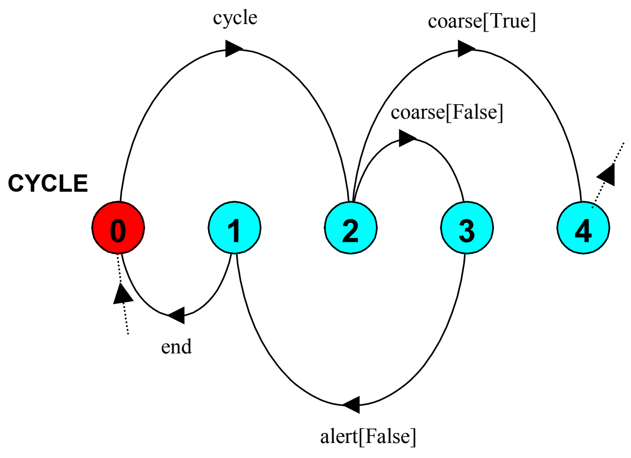

Figure 3 – CYCLE Labeled Transition System

3.2 Graphic Animation

In [1], we described a scheme for associating smooth graphic animation with LTS models. The scheme basically involves mapping model actions to commands that initiate animation activities and mapping animation conditions signaling the end of these activities to guards that enable model actions. The formal foundation for this scheme is Timed Automata [4]. Our animation activities reify the clock variables of Timed Automata. Animations are constructed from sets of Java Beans (called SceneBeans) as directed by an XML document. These beans are of two types: *graphic* beans that are organized into a scene graph structure as dictated by the XML document to depict the visual image of the animation and *behavior* beans that produce time varying values. These values are associated with transformation beans in the scene graph to produce movement in the animation. It is these behavior beans that are started by actions in the LTS and enable actions when they terminate. The relationship between these elements is summarized in Figure 4. The leaf beans of the scene graph draw shapes – rectangles, circles, etc – while intermediate nodes apply transforms such as translation and rotation to the way shapes are drawn.

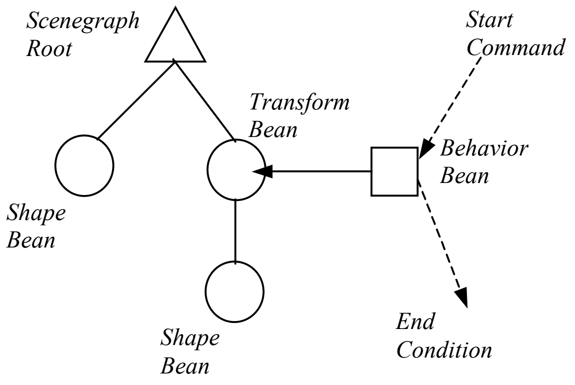

Figure 4 – Graphic Animation elements

The problem posed in applying this animation scheme to the STCA model is – how do we include information about aircraft tracks? This information has been abstracted from the state machine model; however, it is necessary if we wish to include aircraft movement in the animation and more importantly, to permit the replay of recorded tracks leading to STCA conflict alerts in the model. The standard behavior beans in the animation toolkit are simple time varying functions that for example generate the value of a point moving along a straight line as time progresses.

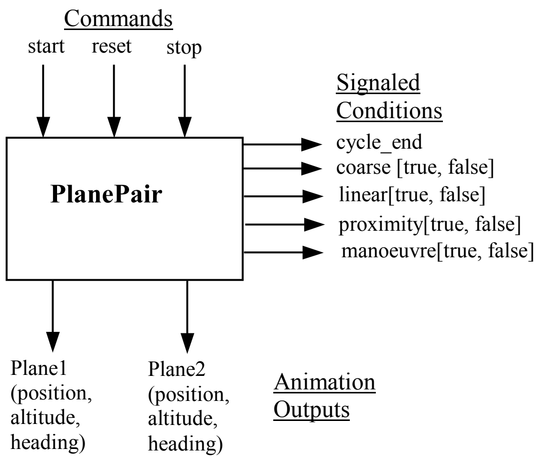

Figure 5 –STCA animation bean

Our solution is to introduce an STCA behavior bean *PlanePair* that takes as a parameter a pair of aircraft tracks. This bean is depicted schematically in Figure 5. The bean accepts the standard commands to start, stop and reset the animation behavior. It produces outputs that can be used to animate the position, altitude and heading of the graphic shapes used to represent aircraft in the animation display (see Figures 6a & 6b). The bean gets its information on the pair of tracks from the XML document that describes the animation. A fragment of the XML declaration of a PlanePair bean is shown below:

```
<behaviour id="planes" algorithm="planePair>

<param name="pointCount" value="49"/>

<param name="plane_one" index="0" value="100.9 -
135.8 -246.0 216.0 13179.0 -34.2 7602S - 15 0 0"/>
<param name="plane_two" index="0" value="80.7 -
127.9 328.0 -128.0 12000.0 0.0 2262S - 15 0 0"/>

<param name="plane_one" index="1" value="100.6 -
135.5 -249.0 218.0 13095.0 -30.1 7602S - 15 0 0"/>
<param name="plane_two" index="1" value="81.19 -
127.94 357.0 -60.0 12000.0 0.0 2262S L 15 0 0"/>
.........
```

The fragment depicts a PlanePair bean declaration that is parameterized with a track pair with 49 data points. Each data point gives the position, altitude, speed and climb rate for each aircraft in the pair. To ensure smooth animation, the bean interpolates the values between these data points as animation time progresses to produce the animation output. The animation output is scaled to map aircraft positions and altitudes into pixel positions on the animation display.

3.3 Combining animation and LTS model

In addition to producing the animation outputs, the PlanePair bean includes boolean functions which implement the coarse filter and fine filter algorithms. These functions are applied to each data point as it is reached in the animation and the result output as signaled conditions. These conditions control the corresponding actions in the LTS model of the STCA. For example, the `coarse[True]` and `coarse[False]` actions of Figure 3 are enabled by the corresponding conditions signaled by the PlanePair bean. In other words, the animation provides the environment of the LTS model. The animation when combined with the LTS model forms a hybrid model of the STCA system that captures both the discrete and continuous behavior of the system.

Figures 6a and 6b are screen shots of the STCA animation display. Figure 6a depicts the situation in which the coarse filter has passed but no fine filter hits have yet been detected. Figure 6b depicts the situation in which the Current Proximity filter (CP) has detected a conflict, the LTS model has confirmed the conflict and the alert signaled. The alert (and filter conditions) are displayed by mapping animation commands that modify the drawing to actions in the LTS model.

The XML document that describes the particular animation of Figure 6 consists of 600 lines of which 150 specify track data for the PlanePair bean. The XML document describing the animation is produced automatically by a translation program that takes as its input a NATS track data file. The program computes the axis labels and the scaling factors that map plane position and altitude to pixel positions. The PlanePair bean consists of around 500 lines of Java code although currently only the coarse and current proximity filter functions are fully implemented. Note that the XML document is specific to a particular set of tracks while the PlanePair bean is the same for all STCA animations.

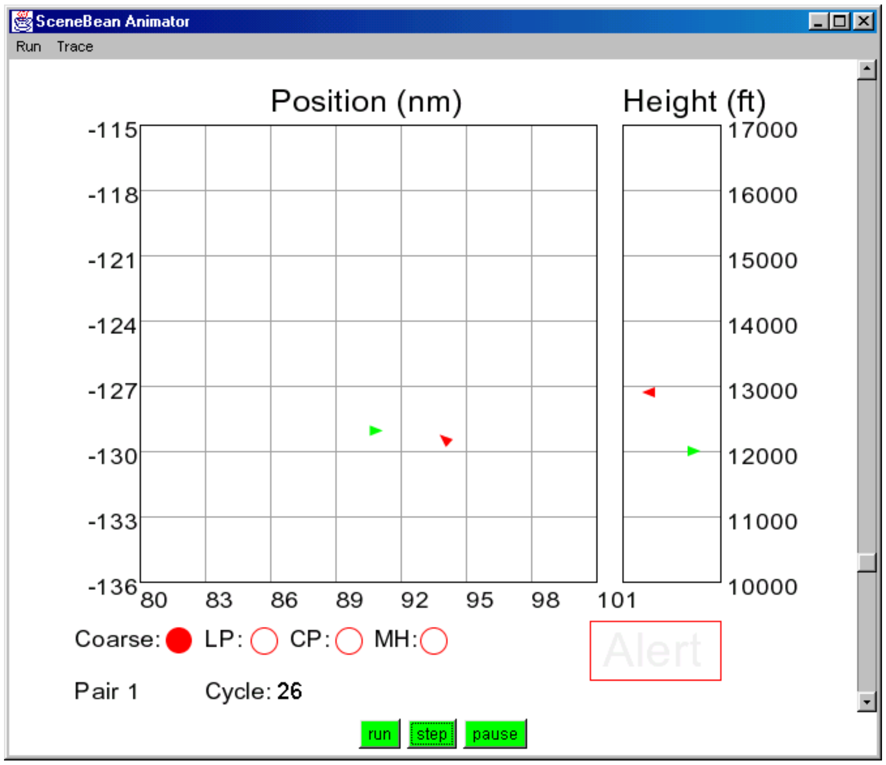

Figure 6a – STCA Animation

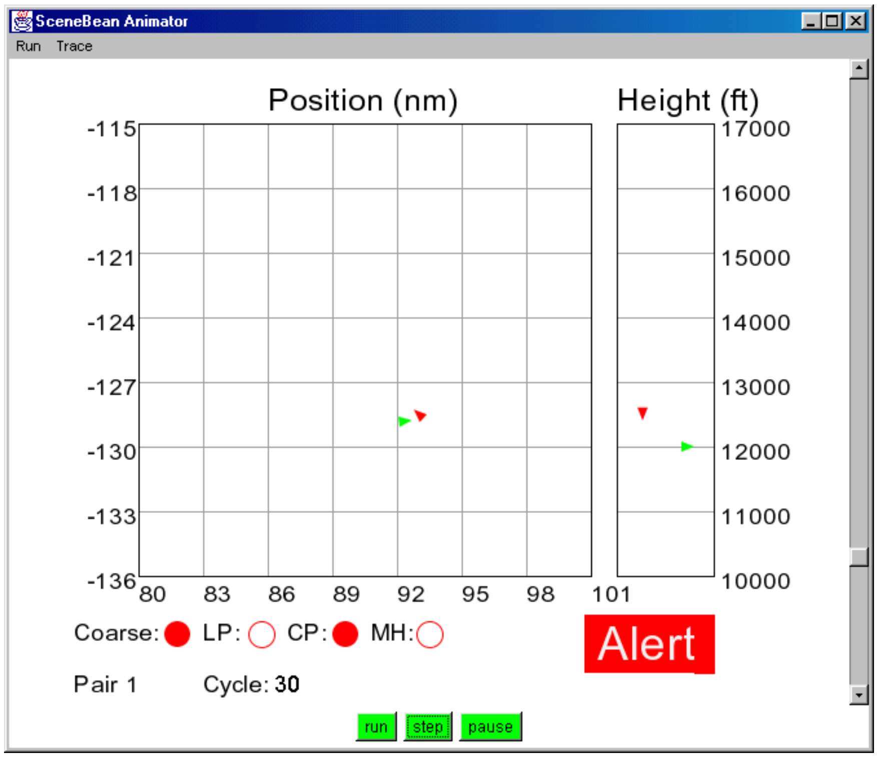

Figure 6b – STCA Animation

4. Related Work

Most behavior modeling and analysis tools provide the ability to execute the model specification as a way of simulating the system being modeled. The output of this simulation is displayed in the context of the specification. For example in SPIN[5], the simulator highlights statements in the Promela specification source as execution proceeds. The Concurrency Factory[6] displays the execution in the context of process diagrams specified in GCCS, a graphical notation for Milner's CCS. UPPAAL[7], a tool based on timed automata, displays simulation results by high-lighting transitions and states of diagrammatic representations of automata produced using the Autograph tool[8]. Graphical animation in these tools thus refers to animation of some graphical representation of the model specification. This is clearly a useful facility in debugging and understanding models – it is a facility provided in the *LTSA* which animates LTS graphs – however, it does not address the problem of communicating in a domain specific way with requirements stakeholders unfamiliar with the modeling formalism.

Some initial work on domain specific visualization is reported by Heitmeyer[9] in the context of the SCR[10] simulator. They use the image of real instrument panels to display the outputs and controls for a simulation of the function of that control panel specified in SCR. The form of animation is similar in scope to that provide by StateMate[11]. StateMate supports animation through a set of predefined widgets that display buttons, lights, dials and graphs. Work on graphical animation of Z specifications using "plug-ins" is referred to in [12]. A graphical animation of the specification of an air traffic control system is referred to but not documented.

The visualization approach we use is perhaps closest to work in the field of program visualization, although the objective of that work is to visualize program execution while ours is to target model visualization at a problem domain. A comprehensive account of the field of Software Visualization may be found in the eponymous book edited by Stasko et al[13].

5. Discussion

Alur, Henziger and Ho[14] define a hybrid automaton as a finite state machine that is equipped with continuous variables. The discrete actions of a system are modeled by explicit transitions in the state machine while continuous activities are modeled by real valued variables whose values change continuously over time. In these terms, the model we construct by combining an LTS with an animation is a hybrid automaton. Alur et al in the same paper present a model-checking procedure for *Linear Hybrid Automata.* In these automata, at each state, the behavior of all variables is governed by linear constraints on the first derivatives, e.g. constant differential equations. Unfortunately, this condition does not hold for aircraft tracks where the position, velocity, altitude etc may vary non-linearly as dictated by the pilot of the aircraft. As a result, we cannot apply the automatic verification procedure for Linear Hybrid Automata to our hybrid model. What therefore is the use of the hybrid model if we cannot model-check? In answering this question, we will first summarize the benefits of the hybrid model that we have identified in the STCA case study and then present our position as to the general use of the hybrid modeling and visualization techniques that we have described in this paper.

5.1 STCA Case Study

As outlined in section 3, the STCA model was constructed in two stages. Firstly, the discrete behavior was modeled and then the graphic animation added to permit visualization. As noted, we could apply exhaustive analysis procedures to the LTS model of the system, and interactively explore the effect of different sequences of significant events such as filter hits and misses. This discrete model when augmented with graphic animation permitted the replay of real radar tracks. In essence, the animation provides the environment for the discrete model and allows the operation of the model to be investigated using real scenarios. The visualization permits these scenarios to be discussed with domain experts. We can then examine the effect of parameter and algorithms changes in relation to these tracks.

Unsurprisingly (and very reassuringly!), the analysis and animation has not indicated any problems in the STCA system since it is a mature system that has been deployed for a number of years. NATS is (obviously) keen to maintain this situation as the system is reviewed and developed. The advantage of our hybrid model in relation to STCA is the superiority of the visualization over the output from the current offline STCA emulator and the ease with which the model can be adapted to test new alert logic and filtering strategies.

5.2 General application

Our motivation in doing this work is the conjecture that the hybrid modeling and visualization techniques that we have applied to STCA are of value early in the development lifecycle of new systems. Our experience with STCA has shown that model development is relatively low cost, requiring a few man-weeks of efforts. The model can be clearly focused on the critical elements in a system and used to explore design alternatives and the feasibility of satisfying requirements. Feedback from the visualizations can help to further develop and enrich the LTS model, and vice versa. Since the hybrid model can only exhibit behaviors already contained in the LTS model (the animation acts as a constraint), the latter can still be used for exhaustive analysis and may produce behaviors and violations not exhibited in the hybrid model. In the STCA case, we were fortunate in having data from real scenarios. In a new system this data might have to be constructed. However, the animation toolkit permits interactive visualizations and for example, we could use the mouse to position aircraft in the STCA animation and generate new scenarios.

The ability to exhaustively analyze systems such as STCA seems remote. Even given the constraints imposed by the physics of flight, the set of trajectories that an aircraft can take in a given region of airspace is very large indeed. As we noted earlier, we cannot simply characterize a trajectory symbolically using equations since the aircraft is controlled by an agent outside the control of the system – the pilot. In this paper, we have discussed a technique that shows the potential to help in the development of this class of system.

References

[1] J. Magee, N. Pryce, D. Giannakopoulou, and J. Kramer, “Graphical Animation of Behavior Models,” presented at ICSE, Limerick, June 2000.

[2] J. Magee, J. Kramer, and D. Giannakopoulou, “Behaviour Analysis of Software Architectures,” presented at 1st Working IFIP Conference on Software Architecture (WICSA1), San Antonio, TX, USA, 22-24 February 1999.

[3] J. Magee and J. Kramer, *Concurrency - State Models & Java Programs*. Chichester: John Wiley & Sons, 1999.

[4] R. Alur and D. L. Dill, “A theory of timed automata.,” *Theoretical Computer Science*, vol. 126, pp. 183-235, 94.

[5] G. J. Holzmann, “The Model Checker SPIN,” *IEEE Transactions on Software Engineering*, vol. 23, pp. 279-295, 97.

[6] R. Cleaveland, P. M. Lewis, S. A. Smolka, and O. Sokolsky, “The Concurrency Factory: A Development Environment for Concurrent Systems,” presented at 8th International Conference on Computer-Aided Verification (CAV'96), New Brunswick, NJ, USA, July/August 1996.

[7] K. G. Larsen, P. Pettersson, and W. Yi, “UPPAAL in a Nutshell,” *Springer International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134-152, 97.

[8] V. Roy and R. de Simone, “Auto/Autograph,” in *Computer-Aided Verification*, R. Kurshan, Ed.: Kluwer Academic Publishers, 1993.

[9] C. Heitmeyer, C. Kirby, and B. Labaw, “The SCR method for Formally Specifying, Verifying and Validating requirements: Tool Support.,” presented at 19th International Conference on Software Engineering (ICSE'97), Boston, Massachussets, USA, May 1997.

[10] R. Bharadwaj and C. Heitmeyer, “Verifying SCR Requirements Specifications Using State Exploration,” presented at 1st ACM Sigplan Workshop on Automated Analysis of Software (AAS'97), Paris, France, January 1997.

[11] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sheman, A. Shtul-Trauring, and M. Trakhtenbrot, “STATEMATE: A Working Environment for the Development of Complex Reactive Systems,” *IEEE Transactions on Software Engineering*, vol. 16, pp. 403-414, 90.

[12] Daniel Hazel, Paul Strooper, Owen Traynor, “Requirements Engineering and Verification using Specification Animation,” http://svrc.it.uq.edu.au/pages/Animation.html, 2000.

[13] J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, *Software Visualization*. Cambridge, Massachusetts: MIT Press, 1998.

[14] T. A. H. Rajeev Alur, Pei-Hsin Ho, “Automatic Symbolic Verification of Embedded Systems,” *IEEE Transactions on Software Engineering*, vol. 22, pp. 181-192, 1996.