Modelling and Verification of Automated Transit Systems, using Timed Automata, Invariants and Simulations

Nancy Lynch *

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

1 Introduction

This paper contains an overview of recent and current work in the M.I.T. Theory of Distributed Systems research group on modelling, verifying and analyzing problems arising in automated transit systems. The problems we consider are inspired by design work in the Personal Rapid Transit (PRT) project at Raytheon (as described to us by Roy Johnson, Steve Spielman and Norm Delisle), and in the California PATH project (as described to us by Shankar Sastry, Datta Godbole and John Lygeros) [7, 6, 13, 3]. Our work is based on the Lynch-Vaandrager timed automaton model [19, 20, 18], extended to include explicit state trajectories and continuous interaction [17]. The formal tools we use include standard techniques for reasoning about concurrent algorithms – invariants, simulations (levels of abstraction) and automaton composition, plus standard methods for reasoning about continuous processes – differential equations.

Our work so far suggests that these methods are capable of providing good results about safety and performance of automated transit systems. The methods support modular system description, verification, analysis and design. They allow a smooth combination of discrete and continuous reasoning in the same framework. They are especially good at handling nondeterminism and approximate information.

2 Background

2.1 Timed Automata and Hybrid I/O Automata

The starting point for our transit project was the Lynch-Vaandrager timed automaton model, which has been used over the past few years to describe and analyze many distributed algorithms and simple real-time systems. The definition of a timed automaton appears in [20, 18]. A variety of proof techniques for timed automata have been developed, including invariant assertions and simulations [20], compositional methods based on shared actions [18, 5, 16], and

* Research supported by ARPA contracts N00014-92-J-4033 and F19628-95-C-0118, NSF grant 922124-CCR, ONR-AFOSR contract F49620-94-1-0199, and U. S. Department of Transportation contract DTRS95G-0001.

temporal logic methods [24]. Applications of the model to asynchronous and timing-based distributed algorithms appear in [22, 15, 12, 11, 16], applications to communication systems appear in [24, 9, 1], and applications to real-time control (trains and gates, steam boiler control) appear in [8, 10].

Briefly, a timed automaton is a labelled transition system having real-valued as well as discrete state components, and allowing continuous state evolution as well as discrete state changes. A timed automaton has a set of states of which a subset are distinguished as start states, a set of actions classified as external (input or output), internal, or time-passage actions, and a set of steps (both discrete and time-passage). As a derived notion, it also has a set of trajectories, which describe evolution of the state over time. A trajectory is obtained by filling in an interval of time solidly with states, so that time-passage steps connect all pairs of states. An execution is an alternating sequence of (possibly trivial) trajectories and discrete steps.

Most of the proofs that have been done using timed automata use invariants (statements that are true about all reachable system states) and simulations (statements of relationships between states of an implementation system and states of a more abstract specification system). Even proofs of timing properties are done in this way; the key idea that makes this work is to build time deadlines (first and last times for certain events to occur) into the automaton state and to involve these deadlines in assertions. Some of the proofs have been automated, using the Larch Prover (LP) [4] and PVS [23]. Other proofs use composition and temporal logic. In these examples, the model works well, yielding clear, unambiguous, and understandable descriptions and proofs.

The work on real-time control suggested to us that some additions to the model would be useful for modelling hybrid systems. In particular, it would be convenient to have trajectories as primitive rather than derived objects; this would allow more direct modelling of physical behavior using physical laws. Also, it would be useful to allow continuous interaction between components via shared continuously-changing variables, in addition to discrete interaction via shared actions; this would allow modelling of systems with continuous controllers or shared clocks, for example. These considerations led us to work on developing a new "hybrid automaton" model.

Some conditions we wanted the new model to satisfy were: (a) We wanted it to be an extension of the timed automaton model, in order to take advantage of earlier results. (b) It should support modular system description, design, verification and analysis (using, for example, composition, abstraction, and system transformation). Modular techniques work very well in reasoning about distributed algorithms, and they should work equally well for hybrid systems. (c) It should be mathematical, not tied to or skewed toward any particular language for programming or specification, nor to any particular proof method or verification system. This would allow us to formulate results quite generally, only introducing restrictions (finite-state, differentiability, integrability, Lipschitz, etc.) where necessary. This generality would make the model flexible enough to be used as the formal basis for many different languages and proof methods. (d) The model should support the effective use of different methods, in particular, those of discrete algorithm analysis and those of control theory, in combination.

Our strategy for obtaining a good hybrid automaton model was to develop the model along with case studies in a particular application. This meant that we needed to choose the "right" application: one that was really hybrid (with lots of interesting continuous and discrete activity), that was simple enough for us to handle yet complicated enough to exercise the theory, and that afforded many opportunities for modular system description. Moreover, in carrying out application case studies, our strategy was to model and analyze many related designs rather than isolated examples; such a coordinated study would permit the formal structure that is useful for the particular application to emerge.

2.2 Automated Transit Systems

The application we chose was Automated Transit Systems (ATS). Among our reasons for this choice were:

1. We originally considered studying air-traffic control. However, air-traffic control is too complex to use in developing basic theory, because it adds the complexities of three-dimensional geometry to those of combining continuous and discrete behavior. Many of the problems arising in the ATS domain seem to be simpler (one-dimensional) versions of problems arising in air-traffic control.
2. The ATS application is important in its own right. There has been a recent surge of interest in ATS, on at least three fronts: Personal Rapid Transit (PRT) systems, in which small public vehicles circulate on tracks under automated control, Intelligent Vehicle Highway Systems (IVHS), in which ordinary cars are augmented with sensor, communication and control devices to allow some automated assistance, and traditional transportation systems, which are now being augmented with some automated control features.
3. ATS is a rich application, appearing to provide the right features to exercise the theory. It contains issues of safety (avoiding crashes, observing maximum speed limits), performance, and comfort. It contains a rich combination of continuous and discrete behavior – a complex real-world system may be controlled by an equally complex distributed computer system. It seems to have a good deal of modularity, for example, system decompositions involving separate vehicles, separate nodes of a distributed computer system, or separate functions. It appears that a system can be described at different levels of abstraction, by considering a derivative-based view versus an explicit function view, or a discrete view versus a continuous view.
4. The ATS area has many similarities with other areas we had studied extensively, in particular, the area of communication systems. Both communication and transit systems involve getting something successfully from a source to a destination, with good throughput and timely arrival.[2] This sim-

[2] There are differences. Messages are not usually thought of as having velocity and acceleration. And it is generally worse to lose a vehicle than it is to lose a message.

ilarity makes it likely that techniques that have been used successfully for communication will carry over to ATS.
5. Engineers working in ATS seem amenable to the use of formal methods, because the area is so safety-critical.

2.3 Our Project

We have begun using timed automata and some extensions to describe and obtain results about typical problems arising in ATS's. The methods we are using include invariants and simulations, composition, and differential equations.

With help from application engineers Johnson, Spielman, Delisle, Sastry, Godbole, and Lygeros, we have been identifying problems arising in ATS's, involving, for example,

1. Attaining and maintaining safe speeds.
2. Attaining and maintaining safe inter-vehicle distances.
3. Implementing typical vehicle *maneuvers*, such as lane changes, merging and diverging at Y-junctions, joining and splitting "platoons" of vehicles, etc.
4. Resolving conflicts among several different planned vehicle maneuvers.
5. Tracking specified vehicle trajectories.
6. Handing off control of vehicles from one computer to a neighboring computer in a distributed computer system.
7. Protecting against catastrophes.
8. Routing.

We are modelling versions of these problems formally and proving various properties (safety, throughput, timely arrival, passenger comfort) of the systems we describe. We consider these problems in the presence of various types of uncertainty, for instance, communication delays and uncertainty in vehicle response. We are trying to identify and use modularity wherever possible. We aim not only for results about the particular problems, but also at a general structured theory for ATS's. Also, as I described above, we are using this work to help us to develop general models for hybrid systems.

The next four sections contain descriptions of some of the particular problems we have modelled. Section 3 contains a study of a simple deceleration maneuver. Section 4 shows two uses of levels of abstraction in reasoning about a simple acceleration maneuver: to relate a derivative view of a system to a function view, and to relate a discrete view to a continuous view. Sections 5 and 6 provide brief summaries of our work on vehicle protection systems and platoon join safety, respectively. The paper closes with a brief conclusion section.

Two other papers in this volume are closely related to this one. In [26], Weinberg, Lynch and Delisle provide a detailed description of our work on vehicle protection systems. And in [17], Lynch, Segala, Vaandrager and Weinberg present the latest version of our general hybrid automaton model, which we call the *hybrid I/O automaton (HIOA) model.*

3 Deceleration

Our first project [21, 25] was the analysis of a simple control maneuver designed to ensure that a vehicle's speed is within a given range $[v_{min}, v_{max}]$ when it reaches a particular track position x_f. The vehicle is assumed to start at position 0 with known velocity $v_s > v_{max}$. A version of this problem was studied earlier by Schneider and co-workers [2].

We considered this problem with uncertain vehicle response and communication delay, and with and without periodic sensor feedback. We proved, using invariants and simulations, that certain example controllers guarantee correct behavior.

3.1 No Feedback

In the simplest version of the problem, there is no feedback from the vehicle to the controller. The controller is allowed to apply a brake at any time, which causes the vehicle to decelerate at some unknown, possibly varying rate in the interval $[a - \epsilon, a]$, where a is a known negative real. The controller can also disengage the brake ("unbrake") at any time. The controller can use only its knowledge of the constants v_s, v_{min}, v_{max} and a to decide when to brake and unbrake. Of course, some restrictions on the constants are needed in order to make such a maneuver possible.

We modelled the vehicle by a single hybrid I/O automaton (HIOA), V, using the model of [17].[3] Its discrete actions are the two inputs, *brake* and *unbrake*. Its state consists of values of the following variables:

$x \in \mathsf{R}$, initially 0
$\dot{x} \in \mathsf{R}$, initially v_s
$\ddot{x} \in \mathsf{R}$, initially 0
$acc \in \mathsf{R}$, initially 0
braking, a Boolean, initially *false*

Here, acc represents the acceleration proposed by the automaton's environment (presumably, a controller) while $\ddot{x}$ represents the actual acceleration. The variables x and $\dot{x}$ represent the position and velocity, respectively. The effects of the discrete inputs are described by the following "code".

brake	*unbrake*
Effect:	Effect:
$braking := true$	$braking := false$
$acc := a$	$acc := 0$
$\ddot{x} :\in [a - \epsilon, a]$	$\ddot{x} := 0$

The trajectories are all the mappings w from left-closed subintervals I of $\mathsf{R}^{\geq 0}$ to states of V such that:

[3] At the time we carried out this project, we actually used a less powerful extension of the timed automaton model, but the newer model works even better.

1. *braking* is unchanged in w.
2. $\ddot{x}$ is an integrable function in w.
3. For all $t \in I$, the following conditions hold in state $w(t)$:
   (a) If $braking = true$ then $\ddot{x} \in [acc - \epsilon, acc]$, otherwise $\ddot{x} = 0$.
   (b) $\dot{x} = w(0).\dot{x} + \int_0^t w(u).\ddot{x} du$.
   (c) $x = w(0).x + \int_0^t w(u).\dot{x} du$.

(The dot after a state is used to indicate state components.) Thus, the *acc* variable is set by the environment (controller), by braking and unbraking. The actual acceleration, velocity and position are determined accordingly: the actual acceleration $\ddot{x}$ is assumed to be in an interval bounded above by *acc* if the brake is on, and otherwise is 0, while the actual velocity and position are determined from $\ddot{x}$ using integration. Our choice of notation for describing V is not important – other notation could be used, as long as it denotes the same HIOA.

Many controllers could be combined with automaton V. We considered a trivial controller that just brakes once, at some time in the interval $[0, t_1]$, then unbrakes once, at some time in the interval $[t_2, t_3]$ after braking. The specific times t_1, t_2 and t_3 were chosen to be as nonrestrictive as possible. We modelled the controller by another HIOA, C. Its discrete actions are the two outputs, *brake* and *unbrake*. It enforces the time bounds t_1, t_2 and t_3 by including deadline variables *last-brake*, *first-unbrake*, and *last-unbrake* in its state, and manipulating them so as to ensure that the *brake* and *unbrake* actions occur at allowed times. That is, initially $last\text{-}brake = t_1$. When *brake* occurs, *first-unbrake* and *last-unbrake* are set to times t_2 and t_3 in the future, respectively. C does not allow time to pass beyond any *last* deadline currently in force, and does not allow an *unbrake* action to occur if its *first* deadline has not yet been reached. The trajectories are trivial – there is no interesting continuous behavior in the controller, so time just passes without changing anything else.

The entire system is modelled formally as the composition of the two HIOA's, V and C. We proved two properties of this composed system, $V \times C$, both involving the behavior of V:

1. If $x = x_f$ then $\dot{x} \in [v_{min}, v_{max}]$.

2. x eventually reaches position x_f.

For example, consider the velocity upper bound, that is, the claim that the velocity at position x_f is at most v_{max}. This claim can be expressed as an invariant, so we wanted to prove it in the usual way for invariants – by induction on the length of an execution. For executions of an HIOA, we take the "length" to be the total number of discrete steps and trajectories. As usual for invariants, we had to strengthen the property so that it could be proved inductively; this involved saying something about states where $x \neq x_f$. By using laws of motion, we came up with the following stronger assertion:

Assertion 3.1 *In all reachable states, if $x \leq x_f$ then $x_f - x \geq \frac{v_{max}^2 - \dot{x}^2}{2a}$.*

This says that there is enough remaining distance to allow the velocity to decrease to v_{max} by position x_f, even if deceleration is the slowest possible.

We proved this strengthened claim using induction. In this inductive proof, the cases involving discrete steps needed only discrete reasoning, while the trajectory cases needed only continuous analysis based on laws of motion. The combined argument implies that the assertion is always true, even with the given combination of continuous and discrete behavior.

For both the velocity lower bound and the "eventuality" property, the key was to show:

Assertion 3.2 *In all reachable states,* $\dot{x} \geq v_{min}$.

Again, this property cannot be proved alone using induction. The key to the proof turned out to be the following claim about the *last-unbrake* deadline while the vehicle is braking:

Assertion 3.3 *In all reachable states, if* $braking = true$ *then* $last\text{-}unbrake \leq now + \frac{v_{min} - \dot{x}}{a - \epsilon}$.

This says that the brake must be turned off before the velocity has a chance to drop below v_{min}, assuming the maximum deceleration $a-\epsilon$, Here, *now* represents the current time. Again, this statement can be proved using induction.

This simple deceleration example already illustrates several aspects of our model and methods: It shows how vehicles and controllers can be modelled using HIOA's and composition, and in particular, how deadline variables can be used to express timing restrictions. It shows some typical correctness conditions – an invariant and an eventuality property – both expressed in terms of the real-world component of the system. It shows how invariants can provide the keys to proofs. Invariants can involve real-valued quantities representing real-world behavior, thus allowing facts about velocities, etc. to be proved by induction; invariants can also involve deadline variables, thus allowing time bounds to be proved by induction.

This example also shows how continuous and discrete reasoning can be combined in a single proof, with formal criteria to ensure that the combination is correct. It illustrates careful handling of uncertainty. Finally, the arguments are general – they don't handle just the apparent worst cases, but all cases at once.

We extended this example slightly to demonstrate some uses of abstraction and composition. Namely, in place of the very nondeterministic automaton C given above, we described the causes of uncertainty in the braking and unbraking times in detail – we supposed that the uncertainty arose entirely from communication delay from a less uncertain controller C' to V. That is, the composition of C' and a "delay buffer" automaton D, $C' \times D$, exhibits behavior that remains within the bounds allowed by the more abstract controller C. Formally, it "implements" C, in the sense of inclusion of external behavior (here, sets of sequences of *brake* and *unbrake* actions, each with an associated time of occurrence).

We showed this inclusion using a *simulation relation* to relate states of $C' \times D$ to states of C. The most important part of the definition of this simulation relation was a set of inequalities involving the deadlines in the two automata. The proof that the relation is a simulation followed the normal pattern for such proofs – it involved showing a correspondence involving start states, one involving discrete steps, and one involving trajectories. Existence of a simulation implies inclusion of external behavior.

External behavior inclusion wasn't quite enough, however. What we really wanted was an exact correspondence between velocity and position values in V, when it is composed with the controller C and when it is composed with the implementation $C' \times D$. But this correspondence can be obtained from the external behavior inclusion result, using basic *projection* and *pasting* results about composition of HIOA's.

3.2 Feedback

We also considered a version of the problem with periodic feedback from the vehicle to the controller, triggering immediate adjustment by the controller of the proposed acceleration. This time, we allowed the controller to set acc to any real value, not just to a fixed value a or 0. As before, the controller's request need not be followed exactly, but only within a tolerance of ϵ.

Our new version of the vehicle automaton V was very similar to the one we used for the no-feedback case. A change is that the new V reports its position x and velocity $\dot{x}$ every time d. In order to express this in terms of an HIOA, we added a *last-sample* deadline component and managed it appropriately. The new V has an $accel(a)$ input action, which causes acc to be set to a. The actual acceleration $\ddot{x}$ is anything in $[acc - \epsilon, acc]$. C performs an $accel$ output immediately after receiving each report.

Now C has more information than before, so it can guarantee more precise velocity bounds. We modelled a controller that initially sets acc to aim so that, if the vehicle followed acc exactly, it would reach velocity exactly v_{max} when $x = x_f$. Since the vehicle might actually decelerate faster than acc, C might observe at any sample point that the vehicle is going slower than expected. In this case, C does not change acc until the velocity actually becomes $\leq v_{max}$. Thereafter, at each sample point, C sets acc to aim to reach v_{max} at exactly the next sample point.

We proved the same two properties for this case as we did for the no-feedback case, but for tighter bounds on the final velocity. The argument again used invariants. For example, consider the argument that in all reachable states, $\dot{x} \geq v_{min}$. Now to prove this by induction, we needed auxiliary statements about what is true between sample points, for example:

Assertion 3.4 *In all reachable states between sample points,*
$\dot{x} + (acc - \epsilon)(last\text{-}sample - now) \geq v_{min}$.

That is, if the current velocity is modified by allowing the minimum acceleration consistent with the current acc, until the next sample point, then the result will still be $\geq v_{min}$. Note the use of the *last-sample* deadline to express the time until the next sample point. This statement is proved using induction.

This example illustrates how our methods can be used to handle more complicated examples, including periodic sampling and control. It shows how to reason about periodic sampling using intermediate invariants involving the *last-sample* deadline: The controller issues control requests to the system at sample times, but can "lose control" of the system's behavior between sample points; the invariants are used to bound how badly the system's performance can degrade between sample points. Again, we handle all cases reliably, not just the apparent worst cases.

4 Levels of Abstraction

Our second project [14] showed how levels of abstraction, one of the most important tools of discrete system analysis, can be used to reason about a simple acceleration maneuver. In this case, the goal is for a vehicle to reach a specified velocity v_f at a specified time t_f in the future. We assumed that the vehicle starts at time 0 with velocity 0. The vehicle reports its velocity to the controller every time d. The controller can send an $accel(a)$ control signal to set $acc := a$ immediately after each sample point. The actual acceleration $\dot{v}$ is anything in the range $[acc - \epsilon, acc]$. The controller we considered aims to reach the goal of v_f at time t_f. That is, it proposes acceleration $\frac{v_f - v}{t_f - now}$, where v is the current velocity.

Using invariants and simulations, we proved bounds on velocity at every point in time. The proofs use levels of abstraction in two ways: relating a derivative view of a system to an explicit function view, and relating a system in which corrections are made at discrete sampling points to a system in which corrections are made continuously. The uncertainty ϵ in the acceleration is integrated throughout the levels.

First, we ignored the discrete sampling and considered a controller that continuously sets acc to the ratio given above, with $\dot{v} \in [acc, acc - \epsilon]$. It was easy to see that the velocity at time t is at most $g(t) = \frac{v_f t}{t_f}$. For the lower bound, by solving the differential equation:

$$\dot{f}(t) = \frac{v_f - f(t)}{t_f - t} - \epsilon,$$

we got a conjectured lower bound of:

$$f(t) = \frac{v_f t}{t_f} + \epsilon(t_f - t)\log(\frac{t_f - t}{t_f})$$

(patched with v_f at t_f). The function f is the result of aiming at (t_f, v_f) and consistently missing low by ϵ.

To prove that f is indeed a lower bound, we used two levels of abstraction. The high level is an HIOA V giving explicit bounds on v. Its state contains only v and now, and the only constraint is that in every reachable state, $v \in [f(now), g(now)]$. The low level is another HIOA D giving bounds on the derivative of v. It keeps acc aiming at (t_f, v_f) and ensures that $\dot{v} \in [acc, acc - \epsilon]$. In a sense, D describes *how* the system is supposed to guarantee the bounds expressed by V.

We showed that D "implements" V, in the sense of inclusion of external behavior (here, the values of v and now). We showed this inclusion using a simulation relation to relate states of D and V. As usual, the proof involved showing a correspondence involving start states, one involving discrete steps, and one involving trajectories. The only interesting case is the one for trajectories. Basically this involved showing that, if the pair (now, v) starts within the region specified by V, the rule used by D does not cause the pair to leave that region. This is in turn proved using standard methods of continuous analysis, expressed formally as invariants involving $\dot{v}$.

Unfortunately, the actual controller does not behave as nicely as D. It only sets acc to aim at (t_f, v_f) at sample points rather than continuously. Between sample points, the value of acc can degrade. In fact, it is not hard to see that v does not necessarily remain above f – the uncertainty introduced by periodic sampling is reflected in a change to the actual behavior produced. Therefore, we had to modify V to reflect the new source of uncertainty. The result was a new V' with a new lower bound f' constructing by aiming not at the "real goal" (t_f, v_f), but at an adjusted goal that depends on d and ϵ, specifically, $(t_f, v_f - \epsilon d)$.

At this point, we could have shown directly (using a simulation relation) that the real system, I, implements V'. However, we found it useful to instead introduce a third level of abstraction, in the form of a modified version D' of D. D' differs from D by having a looser rule for acc: instead of continuously setting acc to aim exactly at (t_f, v_f) it can instead (continuously) set it to point anywhere between (t_f, v_f) and $(t_f, v_f - \epsilon d)$. Thus, D' contains uncertainty in acc, in addition to the ϵ uncertainty in $\dot{v}$. With these simple modifications, we easily modified our proof that D implements V to show that D' implements V'.

Having shown that D' implements V', we were able to forget about V' and just show that I implements D'. Using a transitivity result, this implies that I implements V', as needed.

To show that I implements D', we showed that the identity on all the state components of D' is a simulation relation from I to D'. The key to this proof is the fact that any acc that is set in I is in the range permitted by D'. Note that acc is set to aim at the upper end of its range, (t_f, v_f), at each sample point, but can degrade between sample points. As before, we had to bound the amount of degradation that occurs between sample points. The key claim is that:

Assertion 4.1 *Between sample points,* $acc \geq \frac{v_f - \epsilon(now + d - last\text{-}sample) - v}{t_f - now}$.

This says, roughly speaking, that the value of acc has not degraded too badly if there is still a long time until the next sample point. In particular, at the beginning of a sample interval, $now + d = last\text{-}sample$, so the right-hand side of the inequality simplifies to $\frac{v_f - v}{t_f - now}$, which is exactly the upper end of the range allowed by D'. Also, at the end of a sample interval, $now = last\text{-}sample$, so the right-hand side simplifies to $\frac{v_f - \epsilon d - v}{t_f - now}$, which is exactly the lower end of the range. The complete assertion gives bounds for all the intermediate points as well. This assertion is proved by induction.

This example illustrate more uses of HIOA's and invariants, and the use of *last-sample* deadlines to limit degradation between sample points. Most importantly, it demonstrates two uses of levels of abstraction in reasoning about hybrid control problems: relating a derivative view of a system to a function view, and relating a discrete view to a continuous view. Uncertainties are included throughout, and are handled accurately.

The example also illustrates the useful strategy of specifying the highest-level correctness conditions in terms of an explicitly-specified region of allowed values for the important physical variables. Derivative-based descriptions can be regarded as ways to guarantee that the behavior remains within the high-level regions. For instance, in air-traffic control, the highest-level specification might involve regions in space-time "owned" by particular airplanes. Disjointness of regions then would imply that planes do not collide. The mechanisms for ensuring that individual planes remain within their regions could be reasoned about individually, and separately from consideration of the disjointness of regions.

Note that the bounding functions for the high-level region are obtained using usual methods of continuous analysis – our techniques do not provide any help here. However, our methods do allow systematic checking that the results of the analysis are correct (in particular, that they really capture the worst cases and that they cope correctly with uncertainties).

5 Vehicle Protection Systems

Our third project [26] has been the analysis of automated Vehicle Protection (VP) systems, which are sometimes added to automated Vehicle Operation (VO) systems in order to enforce particular safety constraints. We model both VP and VO systems as HIOA's, and model their combination by composition. Each VP automaton monitors the physical system, using discrete sampling, and checking for "dangerous" conditions. When such conditions occur, the VP triggers an emergency response. For example, a VP might check whether a vehicle's speed is "close" to a specified "overspeed", in order to apply an emergency brake before the overspeed could actually be exceeded. In [26], we analyze both overspeed protection and maintenance of safe separation distance between pairs of vehicles.

This project demonstrates how to model the important interactions between VP and VO systems, using HIOA's and composition. Again, bounding the degradation of physical variables between sample points is a key to the analysis. The project also shows how to compose several VP systems with the same VO system, thereby obtaining the guarantees of all the VP's at once. In this composition, some of the VP's might assume the effects achieved by others. Our work has yielded useful methods for thinking carefully about the design of such systems.

6 Joining Platoons

Finally, our fourth project, just beginning, is the analysis of a "platoon join" maneuver arising in the PATH project [3]. The problem is for cars travelling in a "platoon" to join with another platoon travelling ahead of it in the same lane. The join is accomplished by having the second platoon accelerate to catch up with the first. This introduces the possibility of collisions: We assume that there is some maximum possible deceleration a, the same for all vehicles. If the first platoon suddenly brakes at rate a, and the second platoon is near the first and going faster, then the second platoon will collide with the first, even if the second can react immediately. However, this is considered acceptable by the PATH researchers as long as the relative velocity of the two platoons upon collision is no greater than a small constant v_{allow}.

In [3], a particular controller is described that ensures this relative velocity bound, while allowing the join to be completed as fast as possible and observing passenger comfort limits (expressed by bounds on acceleration and jerk). The controller causes the second platoon to accelerate as fast as possible, subject to safety limits and passenger comfort limits, in order to catch up, and then to decelerate as fast as possible to move into the correct position.

Our goals are to model this system using HIOA's, and to formulate and prove its properties. There are four separate properties to prove: observance of the v_{allow} limit, eventual success in joining platoons, passenger comfort, and optimal join time. Our idea is to use these four separate properties as a basis for decomposing the system and its proof.

So far, we have just considered the safety property – that is, the v_{allow} limit. For this, we are describing a very nondeterministic *safety controller* that just guarantees safety (but not necessarily the other three properties). Our plan is to prove that the safety controller guarantees safety, and then to show that the actual controller implements the safety controller. A bonus is that the safety controller should be reusable for analyzing other maneuvers besides platoon join.

We define a *Platoons* HIOA to model the behavior of the platoons, and allow platoon 1 to be under the control of an arbitrary controller HIOA, C_1. C_1 is unconstrained, subject only to a known maximum deceleration a. The designer's job is to design a safety controller, C_2, for platoon 2 that works with any C_1. That is, the combination of *Platoons*, C_2 and an arbitrary C_1 should guarantee the safety property.

The key to the safety property is an invariant that says that platoon 2's velocity is slow enough, relative to the velocity of platoon 1 and the inter-platoon distance. There are two possibilities, either of which is fine. First:

$$x_1 - x_2 \geq \frac{\dot{x_1}^2 + v_{allow}^2 - \dot{x_2}^2}{2a}$$

This says that enough distance remains to allow platoon 2 to reach v_{allow} by the time a collision occurs, even if platoon 1 decelerates as fast as possible. And second:

$$\dot{x_2} \leq \dot{x_1} + v_{allow}$$

This says that the relative speed is already small enough. The analysis is essentially the same as in [3] (ignoring delays in response), but it is expressed in our invariant style. We can prove that a particular nondeterministic C_2 maintains the disjunction of these two inequalities, and hence guarantees safety.

This example illustrates how our techniques (here, invariants and composition) apply to reasonably complex, realistic systems. It shows how to model a controller that is supposed to work in the presence of unpredictable behavior on the part of some of the real-world entities. Again, we handle all cases reliably, not just the apparent worst cases. Our analysis has given us some insights about the application. For example, we realized that it is important to also model what happens *after* a collision; our techniques appear to be suitable for doing this, but this still remains to be done.

7 Conclusions

We have used hybrid I/O automata to model and verify examples arising in automated transit. We began with very simple deceleration examples, and have progressed to more realistic examples involving vehicle protection systems and platoon join safety. Our methods allow accurate handling of nondeterminism, uncertainties and discrete sampling. All cases are considered, not just the apparent worst cases. The methods support modular system description, verification, analysis and design. They allow a smooth combination of discrete and continuous reasoning, in the same framework.

I have pointed out the key technical features of our approach in various places throughout the paper. The most important of these are: our modelling of all system components (physical world and computer system) as HIOA's; our use of deadline variables to express timing restrictions; our use of composition to describe interactions among components; our statement of correctness conditions in terms of the real world; our extensive use of invariants, including those involving real-valued quantities such as deadlines; our handling of periodic sampling by limiting the degradation of key parameters between sample points; our description of systems at many levels of abstraction; our specification of correctness in terms of regions of allowed values for important physical variables; our use of simulations to show correspondences between different levels of abstraction; and our use of composition to model controllers that work in the presence of unpredictable behavior on the part of some of the system components.

Our preliminary results say that these methods work well to provide useful results about safety and performance of automated transit systems. They have already had some impact on system designers. Our work on ATS modelling has also influenced the development of the basic HIOA model. It remains to use the model and methods to study many more ATS problems, and to integrate the results obtained for all these problems into a coherent theory for automated transit systems.

Acknowledgment: Michael Branicky provided useful information about control theory methods and useful discussions of the platoon join maneuver.

References

1. Doeko Bosscher, Indra Polak, and Frits Vaandrager. Verification of an audio control protocol. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Proceedings of the Third International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems. (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 170–192, Lübeck, Germany, September 1994. Springer-Verlag. Full version available as Report CS-R9445, CWI, Amsterdam, July 1994.
2. Richard A. Brown, Jacob Aizikowitz, Thomas C. Bressoud, Tony Lekas, and Fred Schneider. The trainset railroad simulation. Technical Report TR 93-1329, Cornell University, Department of Computer Science, February 1993.
3. Jonathan Frankel, Luis Alvarez, Roberto Horowitz, and Perry Li. Robust platoon maneuvers for AVHS. Manuscript, Berkeley, November 10, 1994.
4. Stephen J. Garland and John V. Guttag. A guide to LP, the Larch Prover. Research Report 82, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, December 1991.
5. Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy Lynch. Liveness in timed and untimed systems. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming* (21st International Colloquium, ICALP'94, Jerusalem, Israel, July 1994), volume 820 of *Lecture Notes in Computer Science*, pages 166–177. Springer-Verlag, 1994. Full version in MIT/LCS/TR-587.
6. Datta Godbole and John Lygeros. Longitudinal control of the lead car of a platoon. California PATH Technical Memorandum 93-7, Institute of Transportation Studies, University of California, November 1993.
7. Datta N. Godbole, John Lygeros, and Shankar Sastry. Hierarchical hybrid control: A case study. Preliminary report for the California PATH program, Institute of Transportations Studies, University of California, August 1994.
8. Constance Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 120–131, San Juan, Puerto Rico, December 1994. IEEE. Full version in MIT/LCS/TM-511. Later version to appear in C. Heitmeyer and D. Mandrioli, editors *Formal Methods for Real-time Computing*, chapter 4, *Trends in Software* series, John Wiley & Sons, Ltd.
9. Butler W. Lampson, Nancy A. Lynch, and Jørgen F. Søgaard-Andersen. Correctness of at-most-once message delivery protocols. In Richard L. Tenney, Paul D. Amer, and M. Ümit Uyar, editors, *Formal Description Techniques, VI* (Proceedings of the IFIP TC6/WG6.1 Sixth International Conference on Formal Description Techniques - FORTE'93, Boston, MA, October 1993), pages 385–400. North-Holland, 1994.
10. Gunter Leeb and Nancy Lynch. A steam boiler controller. In *Methods for Semantics and Specification*, Schloss, Dagstuhl, Germany, June 1995.
11. Victor Luchangco. Using simulation techniques to prove timing properties. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, June 1995.
12. Victor Luchangco, Ekrem Söylemez, Stephen Garland, and Nancy Lynch. Verifying timing properties of concurrent algorithms. In Dieter Hogrefe and Stefan Leue, editors, *Formal Description Techniques VII: Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques* (FORTE'94, Berne, Switzerland, October 1994), pages 259–273. Chapman and Hall, 1995.

13. John Lygeros and Datta N. Godbole. An interface between continuous and discrete-event controllers for vehicle automation. California PATH Research Report UCB-ITS-PRR-94-12, Institute of Transportations Studies, University of California, April 1994.
14. Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. Manuscript. WWW URL=http://theory.lcs.mit.edu/three-level.html.
15. Nancy Lynch. Simulation techniques for proving properties of real-time systems. In W. P. de Roever, J. W. de Bakker and G. Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives* (REX School/Symposium, Noordwijkerhout, The Netherlands, June 1993), volume 803 of *Lecture Notes in Computer Science*, pages 375–424. Springer-Verlag, 1994.
16. Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1996. To appear.
17. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg. Hybrid I/O automata. This volume.
18. Nancy Lynch and Frits Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing.* To appear. Available now as MIT/LCS/TM-480.c.
19. Nancy Lynch and Frits Vaandrager. Forward and backward simulations for timing-based systems. In J. W. de Bakker et al., editors, *Real-Time: Theory in Practice* (REX Workshop, Mook, The Netherlands, June 1991), volume 600 of *Lecture Notes in Computer Science*, pages 397–446. Springer-Verlag, 1992.
20. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*, 1995. To appear. Available now as MIT/LCS/TM-487.c.
21. Nancy Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In *Second European Workshop on Real-Time and Hybrid Systems*, pages 196–203, Grenoble, France, May/June 1995.
22. Nancy A. Lynch and Hagit Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6(2):121–139, September 1992.
23. Sam Owre, N. Shankar, and John Rushby. User guide for the PVS specification and verification system (draft). Technical report, Computer Science Lab, SRI Intl., Menlo Park, CA, 1993.
24. Jørgen Søgaard-Andersen. *Correctness of Protocols in Distributed Systems.* PhD thesis, Department of Computer Science, Technical University of Denmark, Lyngby, Denmark, December 1993. ID-TR: 1993-131. Full version in MIT/LCS/TR-589, titled "Correctness of Communication Protocols: A Case Study."
25. H. B. Weinberg. Correctness of vehicle control systems: A case study. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, 1995. In progress.
26. H.B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. This volume.

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