Tentative:Towards Fault Tolerance in Distributed Real-time Control

Authors Name/s per 1st Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

Authors Name/s per 2nd Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

***Abstract*—The abstract goes here. DO NOT USE SPECIAL CHARACTERS, SYMBOLS, OR MATH IN YOUR TITLE OR ABSTRACT.**

***Index Terms*—component; formatting; style; styling;**

I. Introduction

Our understanding of real-world computing systems would be incomplete without understanding the effect of failures on such systems. Computers and communication channels failures are captured by failure models such as crashes, omissions, and byzantine behavior, and their effect on computing systems have been well studied in the literature. In fact, the central theme in the distributed computing research is understanding the power and limitations of computing systems under different types of failures. Results in this area provide algorithms for solving canonical problems such as consensus, leader election, and clock synchronization in the presence of certain types of failures, and also establish lower bounds about impossibility of solving those problems with certain resource constraints. Distributed control systems consist not only of computers, but also the physical world with which they interact through sensors and actuators. Thus, to understand the power and limitations of real-world distributed control systems, in this paper, we initiate the study of such systems in the face of computer, sensor, and actuator—failures.

Rough notes: New failure models and their physical manifestations. Sensors and actuators require completely new failure models. Crash, stuck-at, byzantine. Give examples. Also need to consider new failure models for computers, e.g. timing failures. These failures now have physical manifestations, directly challenging safety of the control system.

Failure detection. In distributed computing failure detection is often very hard. E.g., we cannot tell differentiate between a slow and a crashed computer in an asynchronous system. In contrast, failure detection is often easy for actuators. Give example. Timing failures can be detected by runtime environment.

The above suggests new fault-tolerance strategies for distributed control systems. Mantra: Combine fault detection with the model for physical manifestation of failure to avoid bad states.

Our approach (this is very rough): Middleware detects failures and presents a modified world view to the controller. In this world view, faulty agents are "obstacles". Controller reacts to obstacles in the "usual" way.

Case studies. Flocking [1]. Coverage.

Organization:

II. Related Work

Giotto. [2] Simplex [3]. System Simplex. [4] PTIDES. [5] Etherware. [6]

III. Background

Brief description of HIOA. [7] Executions, invariants [8], etc.

This material is likely to be used by both the flocking and the coverage case studies.

Let us describe the physical world model of the two case studies here. The two physical worlds are similar in that they both model the channels (Communications radius: r_{comm}), but they differ in the physics of agents.

We describe the failure models for sensors and actuators here. These failure models are also similar for the two case studies.

Denote the communication radius as r_{comm}, the initial radius to maintain safety as r_{init}, and the safety radius as r_{safety}. To maintain safety, we must have that $r_{comm} > r_{init} > r_{safety}$. Denote the time required to change a node's velocity from maximum to minimum (and vice-versa) as $t_a = \frac{2v_{max}}{a_{max}}$. Denote the time required to travel between the communication radius and the safety radius as $t_v = \frac{r_{comm} - r_{safety}}{2v_{max}}$. Then, $\Delta \leq \min(t_a, t_v)$. Denote the distance covered while changing a node's velocity from one extrema to the other as $r_a = 2v_{max}t_a = \frac{4v_{max}^2}{a_{max}}$. Denote the maximum distance traveled during the delay as $r_d = 2v_{max}\Delta$. (Note that we must fix one of r_{comm} or Δ, as they are functions of one another. We can

automaton PhysicalWorld(d : Real$^+$, r_{comm} : Real$^+$, r_{init} : Real$^+$)
where $d < \Delta$
signature
input send$_i(x, v, u$: Real)
output recv$_{ij}(x, v, u$: Real)

variables
output x_i, x_g, v_i : Real$\forall i \in [N]$;
input u_i : Real$\forall i \in [N]$;
internal $Channel_{ij} : Queue[[< msg : \text{Real}^3, dl : \text{Real}^+ >]] := empty$;
now : Real $:= 0$;
$nbrs : [N] \mapsto 2^{[N]}$;
initially $|x_i - x_j| \geq r_{init} \forall i, j \in [N], i \neq j$;
$x_g >> \max_{i \in N} x_i$;

transitions
input send$_i(x, v, u)$
eff: $Channel_{ij} := append(Channel_{ij}, << x, v, u >, now + \Delta_{delay})$,
$\forall j : |x_i - x_j| \leq r_{comm}$;
$nbrs(i) = nbrs(i) \cup \{j\}$,
$\forall j : |x_i - x_j| \leq r_{comm}$;

output recv$_{ij}(x, v, u)$
pre $\exists! t \in$ Real $:< x, v, u, t > \in Channel_{ij}$
eff $Channel_{ij} = Channel_{ij} - \{< x, v, u, t >\}$;

trajectories
trajdef hold
evolve $\forall i \in [N]$:
$d(x_i) = v_i$;
$d(v_i) = u_i$;
$d(now) = 1$;
stop when $\exists i, j \exists m, t :< x, v, u, t > \in Channel_{ij} \wedge t = now$;

Fig. 1. HIOA Model Physical World

then ensure the given r_{comm} and computed Δ (or vice-versa) satisfy the initial conditions.) Now explicitly, define $r_{init} \equiv r_{safety} + r_a + r_d$, and require $r_{comm} \geq r_{init} + r_d$.

We assume the following of S_0:

1) $\mathbf{x}.x_i > 0$
2) if $j > i \Rightarrow \mathbf{x}.x_j > \mathbf{x}.x_i$
3) $|\mathbf{x}.v_i - \mathbf{x}.v_j| < 2v_{max}$
4) $|\mathbf{x}.x_i - \mathbf{x}.x_j| > r_{init} \qquad \forall i, j \in N$ and $i \neq j$.
5) $u_i = 0$
6) $x_g > x_i$
7) $\Delta \leq \min(t_a, t_v)$

Assume that the network delay between nodes is bounded, such that a message sent by node i is received by all its communication neighbors $j \in N_c(i, s)$ within time d. This assumption immediately gives that all $recv_{ij}$ and $recv_{ji}$ actions will be correct, in that the necessary state information is delivered before the next period. Recall from above that $d < \Delta$, that is, the *communications delay* d is such that messages are delivered before control updates that occur at Δ.

Invariant III.1. $\forall i, j, \mathbf{x}.now_i = \mathbf{x}.now_j = \mathbf{x}.now$.

Proof: This is given by the definition of synchronous communication. ■

Invariant III.2. $\mathcal{I}_{channelSize}(S) = size(\mathbf{x}.Channel_{ij}) \leq 1$

Proof: This holds trivially since all messages will be delivered by time $start + d$, so they will be inserted and removed from the queue in $PhysicalWorld$ before the next Δ. The size when $\mathbf{x}.now = start(\mathbf{x})$ is 1, since all messages are in the channel after the $Send_i$ action occurs at time $start(\mathbf{x})$. The size between $\mathbf{x}.now = start(\mathbf{x})$ and $\mathbf{x}.now = start(\mathbf{x}) + d$ is either 0 or 1, since some messages may have been delivered, but some may not have. The size between $\mathbf{x}.now = start(\mathbf{x}) + d$ and $\mathbf{x}.next$ is 0, since all messages must have been delivered by the stopping conditions and thus no closed trajectory τ could have crossed this boundary. ■

IV. New Flocking

In this section, we study a distributed flocking algorithm under several failure models. Informally, the system consists of N agents starting at arbitrary locations on the real line, where $N \in \mathbb{N}, N > 1$ with the goal of (a) reaching a fixed waypoint $x_g \in \mathbb{R}$, and (b) forming a *flock* or an equispaced formation on the way to x_g. The algorithm we shall study also works when a sequence of waypoints are presented to the agents, but for simplicity of presentation, we fix x_g.

A. Specification

The flocking algorithm is specified by the HIOA Agent$_i$ of Figure 2.

Describe the code. State variables, actions, transitions, trajectories.

The transition $send_i$ occurs at every time $\mathbf{x}.now = next_i$ and adds a message containing physical state measurements of Agent$_i$ to a $Queue$ in PhysicalWorld for each of Agent$_i$'s communication neighbors, to be delivered before $\mathbf{x}.now = d$, such that the state measurements are available for computation at $\mathbf{x}.now = \mathbf{x}.next_i$. Also, the control for Agent$_i$ is set at this transition, based on the measurement values at the time $\mathbf{x}.now - \Delta$. The transition $recv_{ij}$ removes a message containing physical state measurements of Agent$_j$ from a $Queue$ in PhysicalWorld which is for Agent$_i$ and adds it to a *buffer*. Thus, each Agent$_i$ receives knowledge of all nearby neighbors in time.

This state knowledge of neighbors is stored in a *buffer* which maps from the set of indices of nodes, $[N]$ to $(\mathbb{R} \cup \{\perp\})^3$. That is, if an Agent$_i$ has not received a message from Agent$_j$ yet, then $j \notin nbrs_i$, and thus the mapping with j will yield triple of $\{\perp\}$s. Each of the functions $\hat{x}$, $\hat{v}$, and $\hat{u}$ maps from the set of neighbors of Agent$_i$, $nbrs_i$ to triples of physical state variables in $\mathbb{R}$. This achieves the same result as accessing the *buffer*, we should clarify this, as it should be the case that $\hat{x}(j) = buffer(j) = x_j$ from time $\mathbf{x}.next_i - \Delta$. We are using the *buffer* behind the scenes to actually store the message values, and then just use the functions $\hat{x}$, $\hat{v}$, and $\hat{u}$ as a convenient notation. TODO: There is some confusion here–why are we setting the buffer and hat functions separately?

The control, $ctrl(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i)$, is defined as

automaton $\text{Agent}_i(\Delta : \texttt{Real}, i : [N])$ **where** $\Delta > 0$

signature

input $\text{recv}_{ji}(x, v, u : \texttt{Real})$

output $\text{send}_i(x, v, u : \texttt{Real})$

variables

input $x_i, x_g, v_i : \texttt{Real}$;

output $u_i : \texttt{Real} := 0$;

internal $nbrs_i \subseteq [N]$;

$\hat{x}, \hat{v}, \hat{u} : nbrs_i \mapsto \texttt{Real}$;

$now_i : \texttt{Real} := 0$;

$next_i : \texttt{Real} := \Delta$;

$buffer : [N] \mapsto (\texttt{Real} \cup \{\perp\})^3$;

transitions

input $\text{recv}_{ji}(x, v, u)$

eff $buffer[j] =< x, v, u >$;

$nbrs_i = nbrs_i \cup \{j\}$;

$\hat{x}[j] = x$;

$\hat{v}[j] = v$;

$\hat{u}[j] = u$;

output $\text{send}_i(x, v, u)$

pre $now_i = next_i \wedge x = x_i \wedge v = v_i \wedge u = u_i$;

eff $next_i = next_i + \Delta$;

$u_i := ctrl(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i)$;

$nbrs_i = \{\}$;

trajectories

trajdef hold

evolve $d(now_i) = 1$;

stop when $now_i = next_i$;

Fig. 2. HIOA Model of $Agent_i$

$$ctrl\,(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i) = \begin{cases} \text{sgn}\,(x_g - x_i)\, a_{max} & \text{if } nbrs_i = \{\} \\ \text{sgn}\,(\tilde{v})\, a_{max} & \text{elseif } |x_i - \tilde{x}| \leq r_{init} \\ a\tilde{x} - x_i - r_{init}) + b\,(\tilde{v} - v_i) & \text{else} \end{cases}$$

where $\tilde{x} = \hat{x}(\tilde{i})$, $\tilde{v} = \hat{v}(\tilde{i})$, $\tilde{u} = \hat{u}(\tilde{i})$, $\tilde{i} = \underset{j \in nbrs_i}{\text{argmin}}\,(\hat{x}(j) - x_i)$, $a \in \mathbb{R}_+$, and $b \in \mathbb{R}_+$.

Let $\mathcal{A}$ be the HIOA obtained by composing $\text{PhysicalWorld} \| \text{Agent}_1 \| \ldots \text{Agent}_N$. We denote a state of $\mathcal{A}$ by $\mathbf{x}$ and individual state components by $\mathbf{x}.x_i$, $\mathbf{x}.next_i$, etc.

B. Analysis

We would like to show that the *safety property*, that the distance between each node i and j is at least r_{safety} is satisfied. However, before we begin working toward the safety property, we must make some assumptions about communications and prove some simpler invariants.

Lets distinguish between state and physical location, velocity etc. Because the state of each agent includes other variables such as $nbrs_i$, etc.

The next invariant follows from synchronous communication. All nodes receive messages about the state information from time $next - \Delta$ before time $next$. Define the time for the last send with respect to state s as $start(s) = s.next - \Delta$.

Invariant IV.1. $\mathcal{I}_{nbrs}\,(S) =$

$$\begin{cases} \text{if } \mathbf{x}.now - start(\mathbf{x}) > d \\ \text{then } \mathbf{x}.nbrs_i = \mathbf{x}.nbrs(i) \\ \text{elseif } \mathbf{x}.now - start(\mathbf{x}) \leq d \\ \text{then } \mathbf{x}.nbrs_i \subset \mathbf{x}.nbrs(i) \end{cases}$$

Proof: Fix a reachable state s. Expanding $s.now - start(s) \geq d$ to $s.now - s.next + \Delta \geq d$, it is clear that if $s.now \geq start(s) + d$, all messages will have been delivered due to the stopping condition for the $recv_{ij}$ actions to occur in PhysicalWorld, so the set of neighbors in PhysicalWorld for each Agent_i is exactly the same after d time has elapsed. However, before that time, all messages may still be in the channel since no stopping condition has been reached (and all messages were put in the channel at time $start(s)$), or some messages may have been delivered already, so all messages are either in the channel or have been delivered, and thus, the set of neighbors of Agent_i, $s.nbrs_i$, is a subset of the set neighbors in PhysicalWorld, $s.nbrs(i)$. All closed trajectories τ that could violate this are excluded by the stopping condition. ■

Invariant IV.2. The basic property we want to show is that, $\mathbf{x}.x_i > \mathbf{x}.x_j \forall i > j$, but to specify this as a valid statement, we should look at it in more detail, as

$$\mathcal{I}_{noReorder}\,(S) = \begin{cases} \text{if } (\mathbf{x}.now = \mathbf{x}.next_i) \Rightarrow \\ \forall i > j \mathbf{x}.x_i + \mathbf{x}.v_i\Delta + \frac{1}{2}ctrl(\mathbf{x}, i)\Delta^2 > \\ \mathbf{x}.x_j + \mathbf{x}.v_j\Delta + \frac{1}{2}\mathbf{x}.u_j\Delta^2 \\ \text{if } (\mathbf{x}.now \neq \mathbf{x}.next_i) \Rightarrow \\ \forall i > j \mathbf{x}.x_i + \mathbf{x}.v_i\mathbf{x}.t_{Rem} + \frac{1}{2}\mathbf{x}.u_i\mathbf{x}.t_{Rem}^2 > \\ \mathbf{x}.x_j + \mathbf{x}.v_j\mathbf{x}.t_{Rem} + \frac{1}{2}\mathbf{x}.u_j\mathbf{x}.t_{Rem}^2 \end{cases}$$

where $\mathbf{x}.t_{Rem} \equiv (\mathbf{x}.next_i - \mathbf{x}.now)$, $\tau.len = t$, and $ctrl(\mathbf{x}, i) = ctrl(\mathbf{x}.i, \mathbf{x}.x_g, \mathbf{x}.x_i, \mathbf{x}.v_i, \mathbf{x}.u_i, \mathbf{x}.\hat{x}, \mathbf{x}.\hat{v}, \mathbf{x}.\hat{u}, \mathbf{x}.nbrs_i)$.

Proof: The base case is satisfied by assumption on the initial conditions that $\mathbf{x}.x_i - \mathbf{x}.x_j > r_{init}$, so thus, $\mathbf{x}.x_i > \mathbf{x}.x_j + r_{init}$.

The $send_i$ and $recv_{ij}$ actions have been shown to provide state measurements no older than Δ, the sampling period, since the communications delay, d, is less than Δ. These state measurements are used in the computation of the control, so we assume they have arrived in time and delegate this correctness to another invariant. The interesting cases are thus all closed trajectories τ of length $\tau.len = t$. At the control update, we have $\mathbf{x}.x_i + \mathbf{x}.v_i\Delta + \frac{1}{2}ctrl(\mathbf{x}, i)\Delta^2 > \mathbf{x}.x_j + \mathbf{x}.v_j\Delta + \frac{1}{2}\mathbf{x}.u_j\Delta^2$. Then, for all closed trajectories, we have $\mathbf{x}.x_i + \mathbf{x}.v_i(\mathbf{x}.t_{Rem} - t) + \frac{1}{2}\mathbf{x}.u_i(\mathbf{x}.t_{Rem} - t)^2 > \mathbf{x}.x_j + \mathbf{x}.v_j(\mathbf{x}.t_{Rem} - t) + \frac{1}{2}\mathbf{x}.u_j(\mathbf{x}.t_{Rem} - t)^2$. This setup will work since we have set it up accordingly, but it is interesting if we substitute the controls. That is, after manipulation, we see that the control update must satisfy $ctrl(\mathbf{x}, i) > \frac{2(\mathbf{x}.x_j - \mathbf{x}.x_i) + 2(\mathbf{x}.v_j - \mathbf{x}.v_i)\Delta + \mathbf{x}.u_j\Delta^2}{\Delta^2}$. Replacing

each of these quantities with what we know from initial conditions, note that $x_i > x_j \Rightarrow x_j - x_i < 0$, and assume $x_g >> x_i$.

$ctrl(\mathbf{x}, i) > \frac{-2(r_{safety})+2(v_{max})\Delta+\mathbf{x}.u_j\Delta^2}{\Delta^2}$

Then performing the case analysis,

if $nbrs_i = \{\}$

$\operatorname{sgn}(\mathbf{x}.x_g - \mathbf{x}.x_i)\, a_{max} > \frac{-2(r_{safety})+2(v_{max})\Delta+\mathbf{x}.u_j\Delta^2}{\Delta^2}$

$a_{max} > \frac{-2(r_{safety})+2(v_{max})\Delta+\mathbf{x}.u_j\Delta^2}{\Delta^2}$

$a_{max} > \frac{-2(r_{safety})+2(v_{max})\Delta+\mathbf{x}.u_j\Delta^2}{\Delta^2}$

$a_{max} > \frac{-2r_{safety}}{\Delta^2} + \frac{2v_{max}}{\Delta} + \mathbf{x}.u_j$

Here we have a slight problem, as now we are dependent upon what u_j is doing. At best or worst, it is at a_{max} or $-a_{max}$, respectively. If we are working with synchronous communications, we know that $\mathbf{x}.u_j$ will be updated accordingly to maintain safety, such that if it needs to move away, it will apply acceleration in the opposite direction of i's new acceleration. However, when moving to asynchronous we will not have this assumption. At any rate, we can look at the worst case, $u_j = a_{max}$, then

$a_{max} > \frac{-2r_{safety}}{\Delta^2} + \frac{2v_{max}}{\Delta} + a_{max}$

$\frac{r_{safety}}{\Delta} > v_{max}$

Which we know to be true by assumption.

The other cases have to be considered separately, as it is not always the case that the a_{max} control will be applied, for instance, what if i's control is small, whereas j's control is large?

■

Invariant IV.3. The safety invariant is defined as

$$\mathcal{I}_{safety}(S) = \begin{cases} \text{if } (\mathbf{x}.now = \mathbf{x}.next_i) \Rightarrow \\ \forall i\, (\mathbf{x}.x_i - \mathbf{x}.x_j) + (\mathbf{x}.v_i - \mathbf{x}.v_j)\, \Delta + \\ \frac{1}{2}\,(ctrl(\mathbf{x}, i) - \mathbf{x}.u_j)\, \Delta^2 \geq r_{safety} \\ \text{if } (\mathbf{x}.now \neq \mathbf{x}.next_i) \Rightarrow \\ \forall i\, (\mathbf{x}.x_i - \mathbf{x}.x_j) + (\mathbf{x}.v_i - \mathbf{x}.v_j)\, \mathbf{x}.t_{Rem} + \\ \frac{1}{2}\,(\mathbf{x}.u_i - \mathbf{x}.u_j)\, \mathbf{x}.t_{Rem}^2 \geq r_{safety} \end{cases}$$

where $\mathbf{x}.t_{Rem} \equiv (\mathbf{x}.next_i - \mathbf{x}.now)$, $\tau.len = t$, and $ctrl(\mathbf{x}, i) = ctrl(\mathbf{x}.i, \mathbf{x}.x_g, \mathbf{x}.x_i, \mathbf{x}.v_i, \mathbf{x}.u_i, \mathbf{x}.\hat{x}, \mathbf{x}.\hat{v}, \mathbf{x}.\hat{u}, \mathbf{x}.nbrs_i)$.

Proof: There are three possible cases that can arise for the controls at $\mathbf{x}.next_i$, assuming at $start(\mathbf{x}, i)$ the state is okay: There are three interesting cases, two of which are when nodes can now communicate when they could not before, and the other is when nodes are still communicating. If nodes are not communicating, everything is boring as they just apply the go-to-goal control.

- First, when nodes are still more than r_{init} apart, they apply $r_{comm} \geq |x_i - x_j| \geq r_{init} > r_{safety}$ $u_i = a\,(\tilde{x} - x_i - r_{init}) + b\,(\tilde{v} - v_i)$
- Second, when nodes are less than r_{init} apart, they apply the collision avoidance control $r_{comm} \geq r_{init} > |x_i - x_j| > r_{safety}$ $u_i = \operatorname{sgn}(v_j)\, a_{max}$ The only time this situation arises (as in, this is the only time we can recover from it) is as follows: $r_{comm} \geq r_{init} > |x_i - x_j| > r_{init} - r_d > r_{safety}$ that is, the nodes must still be separated by almost the radius required to recover safety, r_{init}, but can tolerate the distance covered over the delay.
- If nodes could previously communicate and still can, it is an interesting case, as they could be in either of the above cases. If $r_{comm} \geq |x_i - x_j| \geq r_{init} > r_{safety}$, then $u_i = a\,(\tilde{x} - x_i - r_{init}) + b\,(\tilde{v} - v_i)$. Otherwise, if $r_{comm} \geq r_{init} > |x_i - x_j| > r_{safety}$ $u_i = \operatorname{sgn}(v_j)\, a_{max}$
- The most boring case is when nodes are still not communications neighbors, they just go towards the goal. $|x_i - x_j| > r_{comm} > r_{init} > r_{safety}$ $u_i = \operatorname{sgn}(x_g - x_i) * a_{max}$

■

Invariant IV.4. Progress

All nodes reach the goal.

How do we specify progress as an invariant? If an agent's velocity is such that it is moving away from the goal, how can we show it is eventually going towards the goal (and then for long enough)? How can this be stated in terms of the current state only? Some statement along the lines of, if agents are in a flock, then the control moves them towards the goal might be what we want. Otherwise, we could have the PhysicalWorld keep a record of all of the velocities over time, and show something like on the average the velocity moves the agents towards the goal. However, this is fairly unclear at this point.

Proof: Show that if $x_g > x_i$, then $v_i > 0 \forall i$ eventually and for long enough time to reach goal. Symmetrically, if $x_g < x_i$, then $v_i < 0 \forall i$ eventually and for long enough time to reach goal. That is, we must show that $\exists t_g$ such that $v_i t_g \geq x_g \forall i$. (Slightly more complicated than this, integral of velocity over time?)

Is it useful to do this proof as a stability proof, showing that eventually, $\forall i, v_i = v_max$, in the direction of the goal?

Or equivalently, showing that $x_i - x_g = 0$, to say that the node eventually reaches the goal by stability. ■

Invariant IV.5. Flocking

This is not possible in general here without a different control.

V. Future Work

VI. Conclusions

References

[1] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory," *Automatic Control, IEEE Transactions on*, vol. 51, no. 3, pp. 401–420, March 2006.

[2] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Proceedings of the IEEE*. Springer-Verlag, 2001, pp. 166–184.

[3] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe on-line control system upgrades," in *Proc. American Control Conference*, Philadelphia, PA, Jun. 1998, pp. 3504–3508.

[4] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," *Real-Time and Embedded Technology and Applications Symposium, IEEE*, vol. 0, pp. 99–107, 2009.

[5] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution strategies for ptides, a programming model for distributed embedded systems," in *to appear in RTAS*, 2009. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/529.html

[6] G. Baliga, S. Graham, L. Sha, and P. Kumar, "Etherware: domain-ware for wireless control networks," May 2004, pp. 155–162.

[7] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.

[8] N. Lynch, "Modelling and verification of automated transit systems, using timed automata, invariants and simulations," in *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996, pp. 449–463.