REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-04-

0483

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	30-JUN-04	Final Report 2001 - 2004

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Real-Time Complex Systems	F49620-01-C-0024

6. AUTHOR(S)

Steve Vestal

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANBIZATION REPORT NUMBER
Honeywell Labs 3660 Technology Drive Minneapolis, MN 55418	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Air Force Office of Scientific Research 801 North Randolph Street Arlington, VA, 22203 NM	

11. SUPPLEMENTAL NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release, distribution unlimited	

Approved for public release, distribution unlimited

13. ABSTRACT (Maximum 200 words)

We developed modified linear hybrid automata models for complex real-time tasks, e.g. tasks that have complex internal behaviors, may interact or synchronize in complex ways, and may have variable or not-fully-known timing characteristics. We report on investigations of partial order methods, approximation methods, abstraction methods, and use of rules to distinguish anomaly-free behaviors, that can be used to improve the tractability of model-checking for such models. We also report on use of insights gained in this work to produce new results for simpler models: anomaly-free preemptive scheduling of repetitive job shops, and a new model and scheduling algorithms for globally asynchronous distributed sample data systems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
real-time, hybrid automata, abstraction, model-checking	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

20 August 2004

Complex Real-Time Systems

2001-2004 Final Report

Contract Number F49620-01-C-0024

Submitted to Air Force Office of Scientific Research

Steve Vestal
Honeywell Labs
3660 Technology Drive
Minneapolis, MN 55418
612-951-7049
steve.vestal@honeywell.com

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	30-JUN-04	Final Report 2001 - 2004

4. TITLE AND SUBTITLE
Real-Time Complex Systems

5. FUNDING NUMBERS
F49620-01-C-0024

6. AUTHOR(S)
Steve Vestal

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Honeywell Labs
3660 Technology Drive
Minneapolis, MN 55418

8. PERFORMING ORGANBIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
801 North Randolph Street
Arlington, VA, 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTAL NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We developed modified linear hybrid automata models for complex real-time tasks, e.g. tasks that have complex internal behaviors, may interact or synchronize in complex ways, and may have variable or not-fully-known timing characteristics. We report on investigations of partial order methods, approximation methods, abstraction methods, and use of rules to distinguish anomaly-free behaviors, that can be used to improve the tractability of model-checking for such models. We also report on use of insights gained in this work to produce new results for simpler models: anomaly-free preemptive scheduling of repetitive job shops, and a new model and scheduling algorithms for globally asynchronous distributed sample data systems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
real-time, hybrid automata, abstraction, model-checking	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			

1. Objectives

The objectives were to enable more dynamic and robust real-time scheduling and analysis of complex large-scale systems in the presence of variable or unpredictable behaviors. We want to support complex task models in which tasks may non-deterministically select among alternative behaviors, in which tasks may use non-trivial protocols to interact with each other, and in which different tasks may have different performance requirements (e.g. hard deadlines versus high average throughputs). We want to develop distributed and highly tractable allocation and scheduling policies that achieve high resource utilizations. We want to provide high assurance that performance requirements will be met in the presence of uncertainty.

We developed and applied new hybrid system models and analytic methods to address these problems. We developed methods to concisely introduce real-time scheduling behaviors into systems of real-time tasks whose behaviors (both temporal and functional) were specified using a hybrid automata notation. We investigated improved model-checking methods for such models, including partial order methods, and the use of rules to distinguish anomalous from anomaly-free behaviors to simplify the model. We developed approximation and hierarchical abstraction methods for more tractable analysis. Using insights gained from this work, we were able to obtain some new results for more traditional models. We developed methods for anomaly-free preemptive scheduling of repetitive job shop problems. We developed a new model and methods for globally asynchronous end-to-end scheduling in heterogeneous systems.

2. Accomplishments/New Findings

We had earlier developed algorithms for improved model-checking of systems of linear hybrid automata. Using these methods, we were able to model-check a problem of real-world size and complexity (a model of the task management code modules of a real-time middleware layer). However, as with all model-checking approaches, these algorithms suffer from combinatorial explosion as the problem size grows. One source of combinatorial explosion occurs when there are many possible transitions out of a system state, where full enumeration will explore all

possible orders in which these transitions can occur. Partial order methods have been very successful in dealing with certain discrete model-checking problems (e.g. the widely-used SPIN tool from Bell Labs). We developed some preliminary partial order methods for linear hybrid automata and prototyped them in our tool, with encouraging results. These methods involve checking to see if, among a set of enabled transitions out of a region, the region reached when they are taken in one order always contains the region reached when they are taken in the other order (in which case the second order need not be explored. We also developed a preliminary approach to prove the correctness of such techniques, involving a kind of "algebra" over sequences of operations during model-checking. However, we were unable to demonstrate truly dramatic benefits (other than for the special case of models that contained many singular guards), or to push through a proof of correctness under the current contract.

After we prototyped our hypothetical partial order method, we were able to solve some problems that were larger than before. Unfortunately, the size of the polyhedra (number of continuous variables and linear inequalities in a region) became large enough that we began to see numeric problems. For example, CPLEX (a very widely-used commercial package) would sometimes determine that the exact same set of inequalities was feasible for one objective but infeasible for another. We conjecture this is due to high degrees of redundancy and degeneracy in the sets of inequalities. We worked with the Institute for Mathematics and its Applications at the University of Minnesota to identify some alternative approaches to deal with this problem (e.g. a modified version of an algorithm that always progresses and hence cannot cycle at degenerate vertices, algorithms that use rational rather than real arithmetic). This problem was presented as a workshop problem at the IMA Mathematical Modeling in Industry Workshop for Graduate Students in August 2004. The report produced by the graduate students who worked on this problem is attached as an appendix.

It has been known for a long time that many multi-resource scheduling algorithms exhibit anomalous behaviors, in the sense that simplifying a problem (e.g. reducing the compute time of some task) may result in a worse schedule. Such schedulers are not robust in the face of run-time variations in behavior. This complicates worst-case analysis and verification of systems. We investigated both execution time anomalies (a deadline is missed when a less-than-worst-case execution time occurs) and release time anomalies (a deadline is missed when a greater-than-minimum inter-release time occurs). We showed that optimal time-triggered preemptive

priority schedules may be anomalous, but then we also showed how any time-triggered preemptive priority schedule (optimal or not) can be converted to an anomaly-free schedule. We showed that anomalous behavior cannot be avoided given arbitrary phase offsets between release times. We made a preliminary identification of a scheduling policy and schedulability analysis algorithm for asynchronous systems with lower-bounded but otherwise nondeterministic release times that is anomaly-free with respect to execution times and has bounded anomalies with respect to release times, but were not able to push through the detailed proof and develop a publishable paper on the current contract. A draft copy of a technical paper that describes these results is attached as an appendix.

A reason for our interest in anomalous scheduling is that, if a system may exhibit anomalous behaviors, then model-checking a hybrid model of that system must explore all possible ranges of compute times and inter-arrival times. We developed a preliminary rule for determining, during model-checking, when a transition can be guaranteed to not introduce anomalous behaviors (when the transition cannot release a task that would preempt another any earlier than it would have otherwise). Such transitions could then be deferred (i.e. ignored in certain system regions), which our preliminary studies suggest could significantly increase the size of problem that could be model-checked. However, we were unable to complete a prototype evaluation or proof of correctness for this method on the current contract.

Several researchers have investigated the use of approximation techniques to improve model-checking speed at the expense of occasional false negative results (if the model checks then it is correct, but some correct models may be incorrectly flagged as erroneous). We prototyped and experimented with several approaches. In our experience, it is very difficult to introduce approximate polyhedra in a way that is both fairly precise (few false positives) yet significantly reduces model-checking time. Any approximate polyhedron must be introduced onto the search-from list in the model checker, and in our exercises this often resulted in more effort rather than less. An approximation so large that it significantly reduced model-checking time was so imprecise that large numbers of correct models were erroneously labeled incorrect. We found only one method that offered modest gain (e.g. a factor of 2 reduction in the number of regions explored with very few false positives). In any reachable discrete state, there are typically numerous polyhedra enumerated. We first grouped these into clusters according to some clustering rule (e.g. their intersection is heuristically likely to be of significant size). We then

generated an approximation that contained each cluster by taking the "earliest" (in some intuitive sense) polyhedron and projecting it forward in time using the normal reach-forward operation so that the result contained the cluster. A method to estimate a set of variable rates needed to do this was used. However, because of the way our model-checker was written (keeping a multiple of a linear constraint so we stored only integer coefficients, rather than rational numbers for each individual coefficient), we were unable to prototype and evaluate an accurate version of this method on the current contract.

Another approach, first explored in discrete process algebras, is to develop an abstract automation that is equivalent (in some sense) to a subset of the concrete automata in a system, then substitute the abstract for that subset and perform analysis on the reduced system. We developed preliminary methods to verify that a hand-developed abstraction (such as a specification of system behavior) is a safe approximation (not necessarily equivalent) to a concrete subsystem. We developed methods not only for hybrid automata, but also for concurrent stochastic automata. These methods were applied within the context of an emerging standard architecture description language for embedded computer systems. A much more complete toolkit of such methods would be needed before this approach became practically applicable, but it appears to offer great promise to improve the development of rigorous specifications as well as improve the tractability of model-checking. These results will be presented at an upcoming IFIP World Congress workshop in Toulouse in August 2004. This paper is attached as an appendix.

All the published research models (of which we are aware) for distributed real-time systems either fall into the time-triggered class (points along a sequence of events are statically assigned a global time of occurrence) or the event-triggered class, including those using traffic regulation or shaping for scheduling and analysis purposes (once a sequence of subtasks arrives, they are executed in order, each released when its successor completes). However, there is a third model that occurs fairly often in practice, a system of periodic tasks that sample each other's outputs. This model is obtained, for example, by starting with a continuous-time model for the activity on each node (as a control engineer would), then converting each node independently to a sampled data system. We first identified this model, and defined a new metric for end-to-end performance (the age of an output is the maximum time elapsed since the inputs on which it is based were sampled), on an internal project looking at possible architectures for next-generation

air transport aircraft (e.g. 7E7). Under contract, we used data obtained from proposed hardware architectures and application software to generate an age scheduling model of real-world size and complexity. We explored methods to multiplex and de-multiplex signals over busses and switched networks. We prototyped the automatic generation of a nonlinear constraint model whose solution would be a feasible schedule that satisfied end-to-end age bounds, and used a commercial tool (AMPL/CONOPTS) to demonstrate that schedules for systems of this size could be tractably solved using this approach. The practical results of this study will be presented at the upcoming SAE World Congress in Reno in November 2004. A draft copy of a paper that gives the technical details is attached as an appendix.

3. Related Publications

"Architecture Specification and Automated Timing and Safety Analysis for a Large Avionics System," to be presented at SAE World Congress, Reno, NV, 3 November 2004.

"Hierarchical Composition and Abstraction in Architecture Models," with Pam Binns, to appear *IFIP Workshop on Architecture Description Languages*, Toulouse, France, 27 August 2004.

"Formalizing Software Architectures for Embedded Systems," with Pam Binns, *First International Workshop on Embedded Software*, Tahoe City, CA, October 2001.

"Formalizing Software Architectures for Embedded Systems," with Pam Binns, *Monterey Workshop* 2001, Monterey, CA, June 2001.

"Modeling and Verification of Real-Time Software Using Extended Linear Hybrid Automata," *NASA Langley Formal Methods Workshop*, June 2000.

"Formal Verification of the MetaH Executive Using Linear Hybrid Automata," *Real-Time Applications Symposium*, June 2000.

4. Interactions/Transitions

The emerging SAE standard Avionics Architecture Description Language (AADL), which is based on our original MetaH language, is expected to be formally issued in Fall 2004. This

standard formally specifies some real-time tasking semantics using hybrid automata concepts developed and demonstrated on this contract. Contact: Bruce Lewis, US Army AMCOM, bruce.lewis@sed.redstone.army.mil

The problems we uncovered in solving for containment among polyhedra in the presence of high degrees of redundancy and degeneracy were presented as a workshop problem at the IMA Mathematical Modeling in Industry Workshop for Graduate Students in August 2004. The resulting technical report is attached as an appendix. Contact: Fadil Santosa, Institute for Mathematics and its Applications, santosa@ima.umn.edu.

We are a subcontractor to the University of Pennsylvania on an NSF program to develop automated testing technologies for hybrid systems. This is an outgrowth of our work on this contract. Contact: Insup Lee, University of Pennsylvania, lee@central.cis.upenn.edu.

5. New Discoveries/Inventions/Patents

No patents were filed under this contract.

6. Honors/Awards

The principle investigator serves as a member of the DUSD(S&T) Avionics Advisory Team. The AAT provided technical consulting to the F/A-22 Raptor and F-35 JSF programs. Contact: Andre Van Tilborg, DUSD (Science & Technology), Andre.VanTilborg@osd.mil

Appendices

A. "New Approaches to Polyhedral Containment Check within a Linear Hybrid Automata Reachability Procedure," technical report prepared by team 5 at the Institute for Mathematics and its Applications 2004 Graduate Workshop, sponsored by the National Science Foundation.

B. "Anomaly Free Real-Time Scheduling," draft publication.

C. "Hierarchical Composition and Abstraction in Architecture Models," to appear ADL Workshop, IFIP World Congress, Toulouse, August 2004.

D. "Real-Time Sampled Signal Flows through Asynchronous Distributed Systems," draft publication.

A Technical Report

NEW APPROACHES TO POLYHEDRAL CONTAINMENT CHECK WITHIN A LINEAR HYBRID AUTOMATA REACHABILITY PROCEDURE [1]

Industry Mentor: Steve Vestal, Honeywell, Inc.

Jessica Conway[2], Ali Khoujmane [3], Gary Kilper[4],
Harun Kurkcu [5], Rochelle Pereira[6], Sonja Petrovic[7]
August 8-18th, 2004.

1 Introduction- Why Polyhedral Containment?

Computer scientists use a variety of specialized models to describe the behavior of an algorithm or computer system over time. Many of these models can be subjected to a kind of analysis called reachability analysis, a special case of model-checking. This analysis answers the question, "Given a model and its initial state, is it possible to reach a second given state from the initial state by any possible behavior of the system?" The model of interest here is a linear hybrid automaton, which is a finite state automaton augmented with a set of continuous variables and a set of rules about how the states of the continuous variables may change as the system passes through a sequence of the discrete locations [1].

Many reachability analyses are at heart an iterative search for a fixed point of sets of reachable states. Given a state known to be reachable, enumerate other states that are reachable from it according to the model. Continue until every newly enumerated state has already been reached. The state space of a linear hybrid automaton is uncountable since it includes the states of continuously varying real-valued variables. but it turns out that

[1]Project 5 of the 2004 IMA Summer Program: Mathematical Modeling in Industry- A Workshop for Graduate Students. This work was supported in part by AFOSR and NSF through IMA

[2]Northwestern University
[3]Texas Tech University
[4]University of Chicago
[5]University of Minnesota
[6]University of Chicago
[7]University of Kentucky

the reachable variable states can all be described by enumerating a set of polyhedra [1]. To determine when the reachability algorithm can terminate, we need to decide if the set of points in a newly enumerated polyhedron is contained in previously enumerated polyhedra. This is often done in practice by maintaining a list of previously enumerated polyhedra, and for each newly enumerated polyhedron checking to see if it is contained in any of these.

An analysis tool developed at Honeywell Labs for reachability analysis of linear hybrid automata used a fairly straight-forward algorithm to check for polyhedral containment[19]. This algorithm required the solution to a set of linear programming problems for each containment test. However, this algorithm exhibited anomalies for a small but significant portion of the polyhedra encountered (100% correct handling of all polyhedra is required for successful model analysis).

For example, CPLEX would sometimes decide a set of constraints was feasible and sometimes infeasible, depending on the goal function used. A second LP solver (a primal/dual algorithm) frequently cycled at degenerate vertices, occasionally indefinitely (the manner in which the polyhedra are constructed tends to introduce many redundant constraints).

The problem presented to the IMA team was to investigate these anomalies and come up with approaches to provide a robust and efficient polyhedral containment test.

Note: We assume that all polyhedra are bounded, as this is the case for the given analysis tool.

2 The Problem: Containment Check

The main objective of this report is to investigate different approaches to check polyhedral containment. The polyhedra are given by sets of inequalities that represent the constraints on the continuous variables in the given hybrid state. The equations represent hyper planes, which form the polyhedron. The directions of the constraint inequalities determine the interior of the polyhedron. Each such inequality can be represented by a row in a constraint matrix.

So the problem can be formulated as follows: given a polyhedron $I = \{x \in R^n : Cx \leq d\}$, decide whether it is contained in a candidate outer polyhedron $O = \{x \in R^n : Ax \leq b\}$. Note also that some of our techniques, in particular the linear programming solvers, assume that $x \geq 0$, but this is

already the case in our problem.

3 Linear Programming Approach

The original version of the reachability analysis tool in [20] solves the containment problem using the following linear programming approach: for each constraint (each facet) of the outer polyhedron, minimize that row of the constraint matrix A as the objective function subject to the inner polyhedron's constraints. The polyhedra are generated in such a way that they are all convex. The idea is to ensure that the inner polyhedron satisfies the constraints of the outer polyhedron, hyperplane by hyperplane; each verification consists of solving a linear programming problem. Given the inner polyhedron $I = \{x|Cx \leq d\}$ and the outer polyhedron $O = \{x|Ax \leq b\}$, label each row of A a_i and the corresponding maximal value b_i for $i = 1, ..., n$, n is the number of constraints in O. Then the following algorithm is applied:

for $i = 1$ to n
  $f =$ max $a_i x$
  subject to $Cx \leq d$
    $x \geq 0$
  if $f > b_i$
    stop; not contained.

If at each step $f \leq b_i$, then the inner polyhedron is indeed contained by the outer polyhedron.

It should be noted that the polyhedra might be built out of a large number of inequalities, so this approach is time consuming. In fact, problems having a few hundred constraints have been encountered.

The most common approaches to solving a linear programming problem (LP) are the Simplex algorithm or a primal-dual algorithm [10]. But there can be problems with these approaches [13]. Now, the inequality systems with which we are primarily concerned are usually overdetermined, so a goodly number of degenerate vertices show up. These vertices are such that a greater number of edges run through them than there are variables to be constrained. Because standard LP solvers work by moving from vertex to vertex along edges until the optimal solution is found, too many edges passing through the same vertex may cause stalling or cycling. It is said that for most practical

applications this is not an obstacle; for our applications, it is.

We wish to minimize or avoid this difficulty. One source of this difficulty is redundancy - extra constraints imposed upon a system of inequalities that are already implicitly there. (It should be noted that the removal of all redundancy, if it were possible, is still not a perfect solution, as there exist degenerate systems that contain no redundancy.) Section 3.1 contains a few approaches or attempted approaches to reduce redundancy. Section 3.2. discusses the least squares primal-dual (LSPD) algorithm; it solves the LP problem in such a way that the degeneracy problem is no longer an obstacle.

3.1 Reducing Redundancy

As stated, reducing redundancy in a system may alleviate problems caused by degeneracy. The following are attempts at such reduction. Also it should be noted that the end of section 5.1.1 points to possible further resources on the topic.

3.1.1 Brute Force Method

We are given the system of $m+1$ inequalities $A\mathbf{x} \leq \mathbf{b}$ in n variables, $\mathbf{x} = (x_1, x_2, ..., x_n)$, where some of the inequalities may be redundant. This approach is explained in [7] and is similar to [20], Let $\mathbf{s}^T$ be the last row of A, A_1 be the remainder of A after the removal of this row, and t be the last element of $\mathbf{b}$, $\mathbf{b_1}$ be the remainder of $\mathbf{b}$ after the reduction. Removing these, we have the reduced system of m inequalities $A_1\mathbf{x} \leq \mathbf{b}_1$; we wish to know if this system implies the last inequality, $\mathbf{s}^T\mathbf{x} \leq t$. To do this, the following linear programming problem is posed:

$$\begin{aligned} \max \quad & s^T x \\ \text{subject to} \quad & A_1 x \leq b_1 \end{aligned}$$

Then if the objective function's optimal value is less then or equal to t, the inequality $\mathbf{s}^T\mathbf{x} \leq t$ is redundant and can be removed.
We repeat this process for each inequality to make the final set of inequalities as redundancy free as possible. Though this method is systematic, it is not very efficient. What's more, as it reduces to an LP problem, it is still itself subject to degeneracy problems.

3.1.2 Other Methods

There were attempts to reduce redundancy in more efficient ways. Redundancy in equality systems can be eliminated using QR decomposition, well explained in [17]. More specifically, one would use a rank-revealing QR decomposition [16] to determine the number of extranneous or linearly dependant constraints we have. [16] is especially nice because it deals with sparse matrices, which we are likely to have. Attempts were made to use this decomposition in different ways to reduce the number of inequality constraints, but nothing came of that. We are unsure why; this method is applicable to equalities, but not inequalities. Also, applying it to degenerate vertices *might* work, except that degeneracy does not imply redundancy.

3.2 Avoiding Stalling at Degenerate Vertices- the LSPD Algorithm

In case that the existence of degenerate vertices cannot be avoided, there is an algorithm that will not stall at such vertices. Namely, it is the least squares primal-dual algorithm (LSPD) [5]. The LSPD completes in finitely many steps and avoids the degeneracy problem by not requiring travel via an edge toward the next vertex. Instead, it guaranteeing travel in the optimal direction towards the best feasible solution. This is done by using a non-negative least squares algorithm [14] as a subroutine.

The least-squares primal-dual algorithm we investigated solves the standard linear programming problem to find $x \in \mathbb{R}^n$ with:

$$\begin{aligned} \min \quad & cx \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

where $b \in \mathbb{R}^m, c^T \in \mathbb{R}^n$ is the cost vector, and $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix of constraints.

3.2.1 The Dual Problem

There is a prefilter to check if the standard problem is infeasible. If this prefilter fails, LSPD instructs us to focus attention on the dual problem. We

now look for a vector $\pi^T \in \mathbb{R}^m$ such that

$$\begin{aligned} \max \quad & \pi b \\ s.t. \quad & \pi A \leq c \end{aligned}$$

Duality yields that an optimal solution for the dual problem produces an optimal solution for the standard problem; if the dual problem is unbounded, the standard problem is infeasible.

3.2.2 NNLS as a Subroutine

To find a solution to the dual problem, we go through a series of iterations. At each stage i, we have a feasible solution to the dual problem π_i and a matrix E_i whose columns are the columns A satisfying a relation dependent on π_i. Here, we look at the non-negative least squares problem [14]

$$\begin{aligned} \min \quad & \|b - E_i x\|^2 \\ s.t. \quad & x \geq 0 \end{aligned}$$

If the minimum of this solution is zero at a vector x_i, it can be shown that π_i is an optimal solution to the dual problem and x_i can be augmented with zeros to provide an optimal solution to the standard problem.

3.2.3 A Better Direction

If the non-negative least squares problem is not zero, Farkas' Lemma provides us with a general direction ρ_i, dependent on E_i and b, in which to find a strictly better solution to the dual problem than π_i. If ρ_i satisfies certain properties relative to A, the dual problem can be shown to be unbounded, and thus, the standard problem is infeasible. Otherwise, we can generate $\pi_{i+1} = \pi_i + t\rho_i$ and begin the iterative process again.

A salient property of the direction of ρ_i is that it need not be along an edge of the dual polyhedron. Unlike the simplex algorithm, ρ_i moves along

the direction of steepest ascent. Since $\|\rho_{i+1}\|^2 < \|\rho_i\|^2$, the dependency of E_i upon ρ dictates that each iteration produces a different matrix E_i. Because each E_i is comprised of columns of A, and since A has finitely many columns, the LPSD algorithm must terminate in finitely many steps. The algorithm avoids degeneracy because it uses NNLS as a subroutine; it took fewer iterations in test problems than the simplex method.

4 Quadratic Programming Approach

We would like to reduce the number of (linear programming) problems to solve. One approach is to formulate the containment check as follows.

Consider the inner and outer polyhedra, given by $I = \{x \in R^n : Cx \leq d\}$ and $O = \{x \in R^n : Ax \leq b\}$.

We show containment does not hold by proving the existence of a point x satisfying $Cx \leq d$ (x is in the inner polyhedra) that satisfies one of the constraints described by $Ax > b$ (x is outside at least one face of the outer polyhedron). Noting that $Ax - b$ measures the "size" of the constraint violation, we formulate the problem as follows, as suggested by Tom Grandine [12]:

$$\begin{aligned} \max\ & y^T(Ax - b) \\ \text{subject to } & Cx \leq d \\ & \|y\|_1 = 1, \\ & y \geq 0. \end{aligned}$$

where y is a new variable vector.

This problem is no longer a linear programming (LP) one, but a *quadratic programming* (QP). Such problems can be solved by QP solvers or by most commercial LP solvers. This objective function will be maximized whenever the largest constraint violation for the outer is multiplied by one, with all the others multiplied by zero. As before, inner is contained within outer whenever the objective function value is ≤ 0. It is not contained whenever a positive objective function value can be obtained. This introduces one new variable for each row in A, but it still solves the problem by solving a single QP problem at the expense of doubling the number of problem variables. Considering this, it will be a good idea whenever A has at least eight rows.

5 New Formulation of Containment Problem

Finally, we propose a new formulation altogether for the polyhedra containment problem. Namely, one can first find all the vertices of the inner polyhedron and check whether they are all contained in the candidate outer polyhedron. Again, boundedness and convexity of our polyhedra guarantee that vertex containment implies polyhedron containment.

The first goal is thus to find all the vertices of the polyhedron given by a set of linear constraints. The process of converting the representation $I = \{x : Ax \leq b\}$ into $I = \{v_1, \ldots, v_n\}$, where v_i are the vertices, is called *vertex enumeration.*

The second goal is to check whether each vertex v_i of the inner polyhedron satisfies the constraints of the candidate outer.

5.1 Vertex Enumeration

We found two algorithms that "efficiently" enumerate vertices. It should be noted that the number of vertices may be combinatorially large with respect to the size of the constraint matrix, but we don't know if this is true for the polehedra seen in Linear Hybrid Automata analysis. These may have special structure we don't understand yet.

5.1.1 The lrs Algorithm

The following outlines a part of the vertex enumeration reverse search algorithm (lrs). It is thoroughly explained in [3]. Recall that a polyhedron can be described by a list of inequalities (H-representation) or by a list of its vertices and extrems rays (V-representation). lrs is a C program that converts a H-representation of a polyhedron to its V-representation, and vise versa. That is, it solves vertex enumeration and convex hull problems.

The lrs is based on the reverse search algorithm. Breifly and informally, the reverse search "rs" algorithm works as follows. Suppose we have a system of m linear inequalities defining a d-dimensional polyhedron in R^d and a vertex of that polyhedron is given by indices of d inequalities whose bounding hyperplanes intersect at the vertex. These indices define a cobasis for the vertex. The complementary set of $m - d$ indices are called a basis. For any given linear objective function, the simplex method generates a path between adjacent bases (or, equivalently, cobases) which are those differing in

one index. The path is terminated when a basis of a vertex maximizing this objective function is found. The path is found by pivoting, which involves iterchanging one of the hyperplanes defining the current cobases with one in the basis. The path chosen from the initial given basis depends on the pivot rule used, which must be finite to avoid cycling. The initial implementation of *rs* used Bland's least subscript rule. If we look at the set of all such paths from all bases of the polyhedron, we get a spanning forest of the graph of adjacent bases of the plyhedron. The root of each subtree of the forest is a basis of an optimum vertex. The reverse search algorithm starts at each root and traces out its subtree in depth first order by reversing the pivot rule.

lrs solves degeneracy by use of the well-known lexicographic pivot selection rule for the simplex method. This rule is defined for a subset of the bases, known as lex-positive. The subgraph of lex-positive bases forms a connected subgraph of the basis graph which covers all vertices of the polyhedron. Furthermore an objective function can be chosen so that the simplex method initiated at any lex-positive optimum basis. If we initiate the reverse search method at this basis and reverse the lexicographic pivot rule we generate a spanning tree of the graph of all lex-positive bases. This is the core of *lrs*.

The main function of *lrs* is to find the vertices and extreme rays of a polyhedron described by a system of linear inequalities. Additional functions of *lrs* are : facet enumeration, computation of voronoi vertices, volume computation, estimation of the output size, and restart capability.

Although *lrs* is a large improvement on *rs*, it is far from an efficient general solution to the vertex enumeration problem. Such a solution should reasonably be required to generate all vertices in time polynomial in the input and ouput size. Currently no such algorithm is known to exist. *lrs* is efficient for vertex enumeration of simple (or near-simple) polyhedra, or dually for facet enumeration of simplicial (or near-simplicial) polyhedra.

Remarks. Note that the polyhedra encountered in our reachability analysis tool are simple (i.e., have no "holes" in them). Also, there may potentially be a combinatorial number of vertices. And, more importantly, this paper includes careful treatment of degeneracy, so it should be looked into for more ideas about solving the degeneracy problem.

5.1.2 The Double Description Algorithm

Another tactic for solving the vertex enumeration problem is the double description method. Here, we enumerate the extreme rays of polyhedra defined by the conditions $Ax \geq 0$ for a real-valued matrix A of dimensions $m \times d$ and $rank(A) = d$. Translations of this method exist to handle the vertex enumeration method [9].

Definitions A *double description (DD) pair* (A, R) is a pair of real-valued matrices satisfying

$$Ax \geq 0 \text{ iff } x = R\lambda \text{ for some } \lambda \geq 0.$$

The *representation matrix* A gives rise to a polyhedral cone,

$$P(A) = \{x \in \mathbb{R}^d : Ax \geq 0\}$$

The *generating matrix* R, too, gives rise to a set,

$$\overline{R} = \{x \in \mathbb{R}^d : x = R\lambda \text{ for some } \lambda \geq 0\}$$

Observe that if (A, R) is a double description pair, $P(A) = \overline{R}$. Minkowski's Theorem for Polyhedral Cones [9] states that given a matrix A of appropriate dimensions, there exists a generating matrix R for which (A, R) is a double description pair. Intuitively, the columns of R correspond to rays generating the cone, $P(A)$. We may ask whether there exists a minimal set of rays generating $P(A)$, or equivalently, whether there exists a *minimal generating matrix* R such that no submatrix generates the cone $P(A)$. These minimal rays or column vectors of R are *extremal rays.*

The Algorithm The algorithm begins with a DD pair (A_0, R_0) in which the rows of A_0 are a subset of the rows of A and R_0 is a minimal matrix for $P(A_0)$. At each stage, a new row of A is added to A_i to yield A_{i+1} and a generating minimal matrix R_{i+1} is constructed. The vectors (extremal rays) used to build R_i are "born" at the $i+1$-stage: the $i+1$ constraint of A is the first constraint which forces these vectors to be extremal rays of $P(A_{i+1})$. This information is stored and helps reduce redundancy. This process continues until all the rows of A have been added to the original submatrix A_i and a final generating matrix R is developed. Conceptually, we build the polyhedron one constraint at a time keeping track of which constraint introduces extremal rays. The algorithmn is efficient in solving degenerate problems where objects are overconstrained.

5.2 Vertex Containment

Once the vertices of the inner polyhedron I are enumerated, we need to check whether they are contained in the candidate outer polyhedron O.

5.2.1 Brute Force Method, Again

There is one obvious way to solve this problem- the brute force way- since we are given the vertices of I and the inequalities that determine the boundary of O. However, there are other ways to check the vertex containment which, if the above "direct" approach proves to be expensive, would be a better solution.

5.2.2 Enumerate both I and O

Suppose it is not very expensive to enumerate vertices. Enumerating the vertices of the outer polyhedron O as well provides the representation $O = \{p_1, \ldots, p_m\}$. Let v be a vertex of I. There exists, indeed, an efficient way of determining whether the given point v lies inside of the polytope determined by vertices of $\{p_1, \ldots, p_m\}$. A standard method that uses a linear programming technique is described in detail by Fukuda [7]. But, as LP solving (of complexity at least $O(n^3)$) is more expensive then evaluating the constraints ($O(n^2)$, so care must be taken in this case. If the LP problem to be solved is small, then this may be a good solution. Recall again that the number of vertices may grow combinatorially in the number of constraints; but we are currently unsure how "big" our polyhedra are.

5.2.3 Projected Containment

Another vertex containment approach would be done in two steps as follows.

- Project the vertices of I onto the facets of O to bring this to lower dimensions;
- Check containment of the projection.

We conjecture that the projection can be done efficiently using some known technique, but this is yet to be investigated.

Further, the point-containment problem has gotten a lot of attention for the two- and three-dimensional cases, as these are the ones most useful for

the computer grapics applications where polyhedral containment checks are frequent. The idea behind these quick point-containment algorithms is simple and can be summarized as follows. First, they calculate the homogeneous coordinate [15] representation of the vertices of the given polyhedron O (this is readily available in computer graphics applications). Next, they determine the signs of the deteminants of the matrices which consist of the vertices' homogeneous coordinates. Finally, the determinant signs obtained from each vertex of O are used as input into a short Boolean expression, whose true-false output indicates whether the polyhedron contains the given point or not.

The trouble with this technique is that, although the Test 4.3.6 and Figure 8 in [15] give the explicit algorithm, it only works for the point-in-polygon test, that is, our projection must be done onto a two-dimensional polygon. However, the background definitions in [15] as well as the last conjecture in [18] suggest that the idea of a projectively invariant point-containment check can be extended to R^n for any $n > 3$. We further conjecture that even though the dimension n may be large, the determinant calculation of a $(n-1) \times (n-1)$ matrix would be quicker than solving a linear programming problem. This would yield a new polyhedral containment check that does not solve large LP or QP problems, but is based on vertex enumeration, projection, determinant calculation, and Boolean expression evaluation. It remains to be shown which of these approaches is more effective.

6 Conclusions

We investigated three methods to handle the polyhedral containment problem. However, due to contractual restrictions, we have been unable to test our various ideas on the actual code to see which best remedies the problem. Our expectations are that at least one of the methods will solve the problem, and one of these positive solutions will be faster than the others. Furthermore, leads for reducing redundancy and dealing with degeneracy problems are worth a longer look. Software packages exist that "solve" the vertex enumeration problems, such as [8]. The efficiency of the vertex enumeration approach cannot be measured until more fundamental research is done on the n-dimensional projectively invariant point containment algorithms.

References

[1] Rajeev Alura, Thomas Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[2] David Avis. *User's Guide for lrs - Version 3.2a*, November 1998. http://cgm.cs.mcgill.ca/ avis/C/USERGUIDE32a.html on August 16, 2004.

[3] David Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. http://cgm.cs.mcgill.ca/ avis/C/lrs.html on August 16, 2004, January 1999.

[4] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8:295–313, 1992.

[5] Earl Barnes, Victoria Chen, Balaji Gopalakrishnan, and Ellis Johnson. A least-squares primal-dual algorithm for solving linear programming problems. *Operations Research Letters*, 30:289–294, 2002.

[6] Komei Fukuda. Is there an efficient way of determining whether a given point q is in the convex hull of a given finite set S of points in R^d?, August 2004. http://www.ifor.math.ethz.ch/staff/fukuda/polyfaq/node22.html.

[7] Komei Fukuda. Is there any efficient algorithm to remove redundant inequalities from a system of linear inequalities, August 2004. http://www.ifor.math.ethz.ch/staff/fukuda/polyfaq/node24.html.

[8] Komei Fukuda and Ichiro Mizukoshi. Vertex enumeration package for convex polytopes and arrangements, version 0.41 beta. Mathematica package, March 1993. http://library.wolfram.com/infocenter/MathSource/440/.

[9] Komei Fukuda and Alain Prodon. Double description method revisited. http://www.cs.mcgill.ca/ fukuda/soft/cdd_home/cdd.html on August 16, 2004.

[10] Saul I. Gass. *Linear Programming: Methods and Applications.* McGraw-Hill, New York, 1985.

[11] Balaji Gopalakrishnan. Personal communication, February 2004.

[12] Thomas Grandine. Personal communication, August 2004.

[13] J.A.J. Hall and K.I.M. McKinnon. The simpest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling. *Mathematical Programming*, 100(1):133–150, 2004.

[14] S.A. Leichner, G.B. Dantzig, and J.W.Davis. A strictly improving linear programming phase I algorithm. *Annals of Operations Research*, 47:409–430, 1993.

[15] Masatoshi Niizeki and Fujio Yamaguchi. Projectively invariant intersection detections for solid modeling. *ACM Transactions on Graphics*, 13(3):277–299, 1994.

[16] Daniel J. Pierce and John G. Lewis. Sparse multifrontal rank revealing qr factorization. *SIAM Journal on Matrix Analysis and Applications*, 18(1):159–180, 1997.

[17] Yousef Saad. *Iterative methods for sparse linear systems.* PWS Pub. Co., Boston, 1996.

[18] Federico Thomas and Carme Torras. A projectively invariant intersection test for polyhedra. *The Visual Computer*, 18(7):405–414, 2002.

[19] Steve Vestal. Modeling and verification of real-time software using extended linear hybrid automomata. Lfm2000: Fifth NASA Langley Formal Methods Workshop, June 2000. http://techreports.larc.nasa.gov/ltrs/PDF/2000/cp/NASA-2000-cp210100.pdf.

[20] Steven Vestal. A new linear hybrid automata reachability procedure. Draft, March 2001.

Anomaly-Free Real-Time Scheduling

Steve Vestal*
steve.vestal@honeywell.com
Honeywell Laboratories
Minneapolis, MN 55418

Draft of 30 June 2004

Abstract

We conjecture that using anomaly-free scheduling may contribute to more robust and easily modified and verified real-time distributed systems. We consider both execution time anomalies (a deadline is missed when a less-than-worst-case execution time occurs) and release time anomalies (a deadline is missed when a greater-than-minimum inter-release time occurs). After discussing this conjecture and defining our scheduling problem, we show that optimal time-triggered preemptive priority schedules may be anomalous. We show how any time-triggered preemptive priority schedule (optimal or not) can be converted to an anomaly-free schedule. We show that anomalous behavior cannot be avoided given arbitrary phase offsets between release times. We give a scheduling policy and schedulability analysis algorithm for asynchronous systems with lower-bounded but otherwise nondeterministic release times that is anomaly-free with respect to execution times and has bounded anomalies with respect to release times.

1 Introduction

It has been known for a long time that many multi-resource scheduling algorithms exhibit anomalous behaviors, in the sense that simplifying the problem can increase response times. For example, a correctly operating system may suddenly start missing deadlines when some tasks start consuming less execution time, e.g. due to data-dependent execution, or due to a software or hardware upgrade. A function may start missing deadlines when another function is turned off or fails. A small change in one execution time may cause significant changes in the order and timing in which other tasks start and complete. Similar anomalies can also occur with sporadic tasks, where an increase in the time between releases may cause missed deadlines.

Eliminating, or at least minimizing, anomalous behavior is useful in practice. Anomaly-free behavior decreases the likelihood that an upgrade to a system will have undesireable timing effects. This property may be useful to help assure time partitioning between applications, e.g. variations or aborts in one application will not introduce timing faults into others. Testing can be simpler and more confident, since worst-case performance can be achieved by testing with only worst-case timing parameters. The general reduction in nondeterministic timing and sequencing behaviors make debugging easier, and may permit reduced instrumentation of systems.

The scheduling approach we discuss assigns intermediate deadlines for each step, then uses earliest deadline first (EDF) scheduling on each processor. Within this context, we present both a globally time-triggered and a globally asynchronous method. We show these methods avoid execution time anomalies in both time-triggered and asynchronous systems. Release time anomalies are absent in time-triggered systems, and are bounded and analyzable in asynchronous systems.

2 Related Work

The existence of execution time anomalies for job shop scheduling was first reported in the literature in a classical paper by Graham[3]. Jackson showed that optimal job shop scheduling does not require the insertion of idle times (unlike nonpreemptive job shop scheduline)[4]. Obserations about and generalizations of Jackson's result are central to our paper.

Andersson and Jonsson have studied preemptive scheduling anomalies in multi-processor systems for tasks that are periodic but without precedence constraints[2]. They also consider dynamic reallocation, which is not normally addressed in an integrated way in the literature on end-to-end scheduling, and which we do not consider in this paper.

The problem we address is a repetitive job shop, often called end-to-end scheduling in the real-time literature[6, 5]. Much of this literature implicitly addresses what we call release time anomalies. The typical reasons for this are temporal nondeterminism in external release times, and the release of an intermediate step when its predecessor completes. Methods of scheduling and analyzing such systems implicitly bound any resulting anomalous behavior. As we discuss shortly, unexpectedly early completion of a step is another reason why the release time of its succes-

*This work was supported by the US Air Force Office of Scientific Research under contract F49620-01-C-0024.

sor may vary. Execution time anomalies are (to our knowledge) almost never explicitly discussed in the literature.

3 Repetitive Job Shop

Our results are primarily based on earlier job shop scheduling work, but our intended application is real-time computer systems. We choose to satisfy no one by using a mixture of terminology from the two fields in our definitions.

We define a repetitive real-time job shop as follows. A job is a finite sequence of steps. A step is bound to one of a finite set of processors in the problem statement, which is the processor responsible for executing that step. A preemptive schedule assigns one or more intervals of time on that processor to a step, where the sum of the lengths of the intervals equals a specified step execution time. The last point in the last execution interval in a schedule is called the completion time for that step. Each job has a release time, before which the first step of that job may not be executed (the start of that step's first execution interval in a schedule cannot occur before the job release time). Subsequent steps in a job may not begin execution until their predecessor step completes. A schedule is feasible if the final step in every job completes before a specified job deadline, otherwise the schedule is said to be infeasible. A deadline is specified as a value added to the release time, i.e. the deadline occurs a fixed interval of time after the release time. A task is a possibly infinite sequence of jobs, where the release time for each job does not precede the deadline of the preceeding job in a task sequence.

4 Scheduling

A preemptive priority schedule uses a priority relation $S_i \prec S_j$ between every pair of steps that might ever be simultaneously executable on the same processor according to the precedence rules of the previous paragraph. At any point in time at which a step is being executed by a processor, it is always the step that preceeds all other ready steps in the priority order. That is, whenever a processor is executing, it is always executing the highest priority ready step.

Theorem: For every feasible schedule there is a preemptive priority schedule in which the completion time for every step is no greater than in the original schedule.

Proof: Prioritize steps according to their completion times in the original schedule (an earliest completion first priority relation). Now transform the original schedule as follows. If any S_i completes earlier than some S_j, but there is an interval during which both are ready but S_j is executing, then have S_i rather than S_j execute in that interval. S_i will complete earlier by just that duration; in the vacant interval created following S_i by this earlier completion, execute S_j. That is, swap the execution intervals of S_i and S_j that are not in priority order. This transformation can be repeated until a preemptive priority schedule is obtained, once in which no step completes later than in the original schedule. □

The preceding definition of a preemptive priority schedule permits inserted idle times, i.e. the processor need not be executing when steps are ready. We will call these lazy intervals. A schedule that has no lazy intervals is called a work conserving schedule. In a work conservaing schedule, no processor is idle when there are ready steps that it could be executing.

Theorem (Jackson): For every feasible earliest completion preemptive priority schedule, every step will complete at least as soon in a work conserving earliest completion preemptive priority schedule as in the original (possibly lazy) schedule.

Proof: Find the earliest lazy interval, resolving ties arbitrarily. Start executing the highest ready priority step in this interval rather than deferring it. Since this is the highest priority step, it must be the one executed in the interval immediately following the lazy interval. This transformation slides the executuion interval earlier in the schedule to fill the lazy interval, which is to say it swaps the lazy interval for this next execution interval. Either this step then completes earlier (this step completes at the end of this interval), or this transformation has no effect on any completion time. This transformation may create a new lazy interval on the processor hosting the successor step, but this lazy interval occurs later than the original one. This transformation can be repeated to move lazy intervals later and later in the schedule, until they reach a point at which they are no longer lazy because there are no ready steps to be executed. Such points always exist because a feasible schedule never loads any processor beyond 100% utilization. □

5 Anomalous Scheduling

We say a given scheduling algorithm (an algorithm that accepts a repetitive real-time job shop problem and produces a schedule) is anomaly-free with respect to execution time if, for any feasibly scheduled problem, the execution times for any subset of the steps can be reduced and the scheduling algorithm will also feasibly execute this reduced-load problem.

An optimal preemptive priority schedule for a classical job shop (which is a special case of our repetitive job shop) is one that minimizes the maximum completion time across all jobs (called the makespan). As the following theorem shows, there may be priority assignments that achieve optimality but are not anomaly-free with respect to execution time (which we write as "anomaly-free w.r.t. execution time").

Theorem: An optimal work conserving preemptive priority schedule may be anomalous w.r.t. execution time.

Proof: Figure 1 shows on the left an optimal schedule

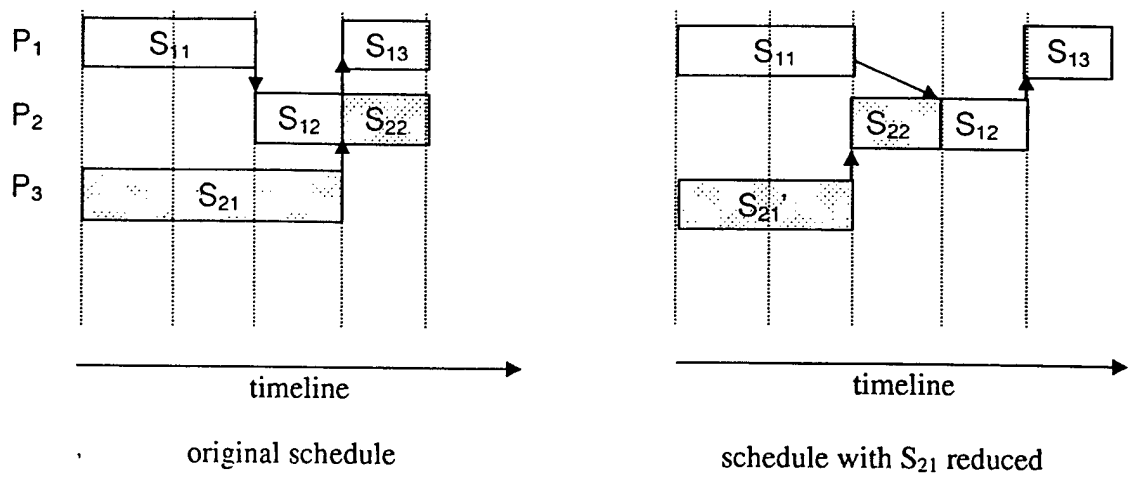

Figure 1: Optimal but Anonomalous Preemptive Priority Schedule

for a two job problem. The schedule is obviously optimal because the makespan is equal to the sum of the step execution times of either job (i.e. the makespan would not change even if either job were executed alone with no contention at all). The schedule on the right is obtained when the execution time for S_{21} is reduced, but work-conserving preemptive priority scheduling is applied with the same priority ordering $S_{22} \prec S_{12}$. The makespan increases. Note the priority ordering in the original solution was not a completion time priority ordering. □

We say a scheduling algorithm is anomaly-free with respect to release times if, for any feasibly scheduled problem, the intervals between the release times of pairs of sequential jobs can be increased and the scheduling algorithm will always feasibly execute this reduced-load problem.

Theorem: There exists no scheduling algorithm that is anomaly-free with respect to release times.

Proof: There exist problems that are feasibly scheduled but where increasing an inter-release time creates an interval in which some processor would need to execute at over 100% utilization to meet deadlines. Figure 2 shows such a problem. The original schedule was feasible because release times permitted an efficient interleaving of step executions, but shifting the release of a step in the modified problem now overloads the processor between two step release times and deadlines regardless of how it is scheduled. □

6 Anomaly-Free Scheduling

Any preemptive priority schedule can be easily modified so that it is anomaly-free by rearranging priorities so they are in order of completion time. This post-processing step can easily (and we suggest should) be applied to any algorithm that searches for optimal priority assignments but has not been guaranted to produce priorities in order of completion times. The following theorem justifies this.

Theorem: For every feasible schedule, a work conserving earliest completion preemptive priority schedule is anomaly-free w.r.t. execution times.

Proof: Construct the work-conserving earliest completion preemptive priority schedule for the original problem (the one with worst-case step execution times). For a step that completes earlier than originally specified, make the unused portion of the execution intervals in the original schedule into lazy intervals, so that no other completion times change. By a preceding theorem, these lazy intervals can be removed to obtain a work-conserving earliest completion preemptive priority schedule in which no step completes later than in the original schedule. □

We can eliminate the possibility of anomalies w.r.t. release times by fixing them, which is what happens in globally time-triggered systems. In traditional time-triggered systems, a fixed global release time is assigned to every step, i.e. run-time traffic regulators are used. We can relax this somewhat and still achieve anomaly-free behavior. We can use a work-conserving preemptive priority scheduler on every processor and still be anomaly-free w.r.t. execution times, if we use the following method for assigning priorities. Run-time traffic regulation is no longer required, which simplifies the implementation somewhat.

Instead of an intermediate release time, we instead assign an intermediate deadline to each step. The deadline of the final step in a job equals the job deadline. These do not have to be checked or enforced at run-time, correct operation could be assured using off-line schedulability analysis. Priorities are then assigned in order of these deadlines. Note that if the task shop is strictly periodic and repeats after some fi-

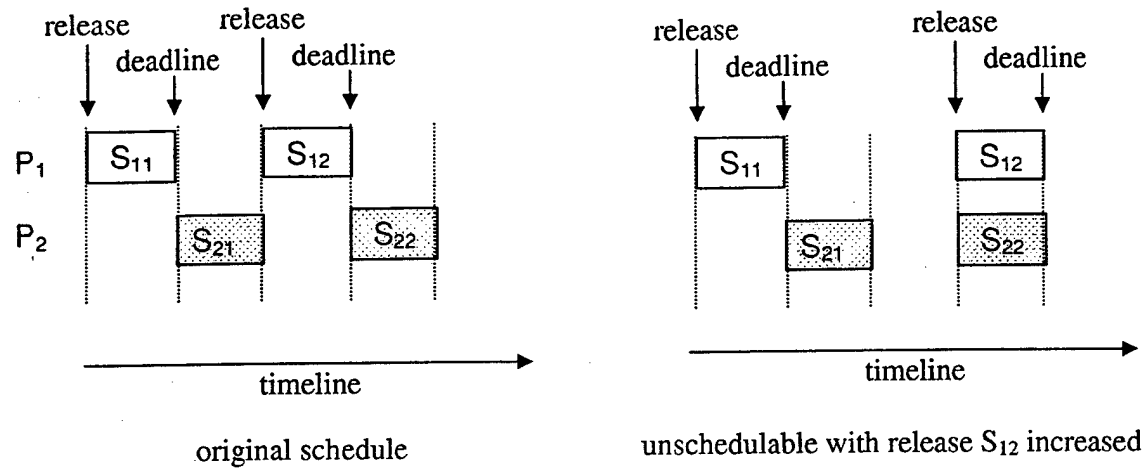

Figure 2: Increased Inter-Release Time Makes Problem Unschedulable

nite hyperperiod, then this can be accomplished using an off-line assignment of fixed priorities to each step instance. Different steps within a task and different instances of the same step might have different priorities, so this priority assignment is closer to classical earliest deadline first than classical preemptive fixed priority scheduling. This is really a class of algorithms because we do not specify a particular algorithm for assigning intermediate deadlines, a subject to which we return in the final section.

To see that this is anomaly-free w.r.t. execution times, first consider the schedule obtained when every step executes for its specified (worst-case) execution time. In every case where two steps are ready at the same time, the step that completes first is the step having earliest deadline. That is, the priority assignment is consistent with completion times as well as deadlines. By a preceding theorem, such preemptive priority schedules are anomaly-free w.r.t. execution times.

We define an asynchronous scheduling model as follows. For each task, define a minimum job inter-release time and a deadline, where the deadline is never greater than the inter-release time. The first job of a task can arrive at any time, which becomes its release time (unknown until the job arrives). Subsequent jobs can arrive any time at or after the release time of the preceding job plus the minimum inter-release time. The deadline for a job occurs at a specified interval of time following the release time.

We define a class of scheduling algorithms for the asynchronous model that is anomaly-free w.r.t. execution times as follows. When a job arrives at time T, we assign intermediate deadlines to every step in that job, where the deadline for the final step is the job deadline. At each processor, use earliest deadline first scheduling. Processor clocks need not be globally synchronized to implement this algorithm, but we do require that one processor be able to observe the local clock of another (with bounded error) at certain synchronization events. Given this capability, the intermediate release times and deadlines for the steps of a job can be adjusted (with bounded error) by the difference of the processor clocks as part of the intermediate step release hand-shaking protocol.

Theorem: The preceding class of algorithms is anomaly-free w.r.t. execution times. By this we mean, if the algorithm produces a feasible schedule for all possible allowed job release times, then it produces a feasible schedule for all possible allowed job release times when one or more execution times are reduced.

Proof: For each fixed pattern of job release times, the pattern of deadlines is fixed. By the same reasoning as above, the completion times for this pattern can only decrease after reductions in execution times. Every individual pattern is thus anomaly-free, so the schedule is anomaly-free for all possible patterns.

7 Future Work

We know the final class of algorithms discussed cannot be anomaly-free w.r.t. release times because completion times depend on the relative phasings of job releases. A schedulability algorithm that bounds worst-case response times for all possible phasings also bounds the anomalies w.r.t. release times. We conjecture that the schedulability analysis algorithms of Spuri can be used for this purpose[5]. We conjecture the bounds on anomalies w.r.t. release times implied by these algorithms also bound any anomalies w.r.t. execution times.

The conjecture stated at the beginning of this paper, that anomaly-free scheduling may contribute to more robust and easily modified and verified systems, has intuitive appeal. Determinism is widely-acknowledged to be highly desireable in safety-critical systems, and adds theoretical complexity to demonstrations of equivalence and compliance between for-

mal models. Totally determistic and anomaly-free behavior isn't possible in distributed systems, but benefits may accrue if these effects can be constrained. Our conjecture is perhaps more pragmatic than theoretical and needs to be assessed by experience.

Anomalous behaviors can result in changes in the order in which the start and end of task executions occur. Nondeterminism in event orderings may complicate debugging and verification. A better formalization and understanding of anomalous behavior with respect to event order might be of use in problem areas such as instrumentation, visualization, syncyhronization and verification of distributed real-time systems. For example, the results noted in this paper imply that if a partial order is enforced on step deadlines, then a corresponding partial order will apply to step completion times.

If we try to compare the theoretical efficiency of a globally time-triggered versus a globally asynchronous solution, we need to be careful about the actual timing requirements. For a sampled data system that periodically samples a continuous input signal, we are presumably free to pick the exact times of sampling and output (as long as they are periodic). The scheduling algorithm is free to slide these back and forth to interleave different jobs efficiently in time. Figure 2 showed an example that was unschedulable using the asynchronous model but feasibly scheduled using a time-triggered model. However, if the application requires bounded response times to external real-world events, then a time-triggered solution essentially polls the external environment, and any polling latency must be added to the end-to-end latency of the time-triggered model. Taking these factors into consideration, it is an open question whether one approach is inherently potentially more efficient than the other for this problem.

We have not discussed the problem of picking good intermediate deadlines in time-triggered systems, which is presumably as difficult (in both theory and practice) as job shop scheduling. We have had some success applying a kind of temporal load balancing to fairly large problems abstracted from actual avionics systems (e.g. thousands of tasks and messages hosted on dozens of processors and busses). However, this has not yet been not generalized to the problem model presented here[1].

Our real-time task shop model assumes each job is a linear sequence of steps. In practice, fan-in and fan-out and feed-back control loops will appear. The latter might be dealt with by breaking each loop and imposing a suitable end-to-end deadline for the associated internal state update, so that a solution for the resulting acyclic graph might be sufficient for most practical purposes.

We conjecture that global end-to-end scheduling algorithms, such as earliest job deadline first, or global least laxity (laxity computed using the sum of remaining step execution time and the final job deadline), are also anomalous. It is an interesting open question whether an anomaly-free scheduling algorithm exists that does not make use of either intermediate release time or intermediate deadline assignments.

References

[1] Robert Allen, Dennis Cornhill, Bruce Lewis and Steve Vestal, "Using an Architecture Description Language for Quantitative Analysis of Real-Time Systems," *Third International Workshop on Software Performance*, Rome, Italy, July 2002.

[2] Björn Andersson and Jan Jonsson, *Preemptive Multiprocessor Scheduling Anomalies*, Chalmers University of Technology Technical Report 01-9, September 2001.

[3] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal of Applied Mathematics*, V17, n2, March 1969.

[4] J. R. Jackson, *Scheduling a Production Line to Minimize Maximum Tardiness*, Research Report 43, Management Sciences Research Project, UCLA, 1955.

[5] Marco Spuri, *Analysis of Deadline Scheduled Real-Time Systems*, Rapport de recherche n. 2772, Institut National de Recherche en Informatique et en Automatique (INRIA), Rocquencourt, France, January 1996.

[6] Jun Sun and Jane Liu, "Synchronization Protocols in Distributed Real-Time Systems," *Proceedings of the* 16th *ICDS*, 1996.

HIERARCHICAL COMPOSITION AND ABSTRACTION IN ARCHITECTURE MODELS

Pam Binns and Steve Vestal
Honeywell Laboratories
Minneapolis, MN, USA
{*pam.binns,steve.vestal*}*@honeywell.com* *

Abstract We present a compositional approach to generate linear hybrid automata timing models, and Markovian stochastic automata safety models, from an architecture specification. Formal models declared for components are composed to form an overall model for the system, where the composition rules depend on the semantics of the architecture specification. We further allow abstract models to be specified for a subsystem of components, where the abstract model may be substituted for the concrete model of that subsystem when composing the overall system model. We assume both abstract and concrete models are given, we address the problem of verifying that the abstractions yield safe if approximate results. An abstract model may be viewed as a formal subsystem specification used for both conformance checking and improving the tractability of system analysis.

Keywords: architecture description language, formal specification, hybrid automata, stochastic processes, schedulability modeling, reliability modeling, system safety

1. Introduction

Given a specification for the architecture of an embedded computer system, we want to generate and analyze formal models of system behavior. In this paper we discuss the generation and analysis of timing and safety models from specifications written in the SAE standard Architecture Analysis and Design Language (AADL) and its original research basis, MetaH[AADL 2004, MetaH 2000].

An architecture is often informally described as an assembly of connected components. Overall system behavior is determined by the interactions between components according to the way they are connected, which is to say system behavior is defined as a composition of the behaviors of its components. We will associate formal models with individual components in a specification. The formal models for a complete system are defined as compositions of the

*This work was supported by the US Air Force Office of Scientific Research under contract number F49620-97-C-0008.

individual component models. In this paper, we use a type of hybrid automaton to specify real-time component behaviors, and a type of stochastic automaton to specify component fault and error behaviors.

Architectures are specified hierarchically. Every component may have an internal implementation that may itself be specified as a set of connected subcomponents. Given a component that has an internal architecture, a formal model for that component can be generated by composing the models for its subcomponents. We call this the concrete model for that component. We may also directly associate an abstract model with a component that is intended to be a safe approximation for the concrete model. When generating a system model from an architecture specification, we thus have a choice for each component whether to use its concrete model or its abstract model. A different choice can be made for different components at different levels of the design hierarchy, so that a fairly large set of mixed-fidelity models is possible. Hierarchical abstraction can both improve understandability and enable tractable analysis for large and complex specifications.

We assume both concrete and abstract models are given, e.g. hand-developed. Our focus is on verifying that analyses performed when abstract subsystem models are substituted for concrete subsystem models are safe in some sense with respect to analyses of the fully detailed concrete models. In the case of our timing models, we show how to verify that classical periodic tasks are conservative approximations for hybrid automata used in the AADL standard to define thread semantics, or hybrid automata that model reusable middleware. In the case of our safety models, we explore the relationship between abstract and concrete stochastic automata models. We expect the effort required to develop pairs of abstract and concrete models to be justified by high degrees of reuse; and that many pairs of abstract and concrete models will be based on common and easily modified design patterns. An abstract model may be viewed as a formal specification that is also usable to improve the tractability of analysis.

2. Related Work

We borrow one of the fundamental ideas of process algebra[Milner 1989]: show that a large and complicated subsystem model can be replaced by a smaller and simpler subsystem model when performing overall system analysis. We permit the smaller simpler model to be an approximate abstraction rather than requiring some notion of equivalence. We deal with hybrid and stochastic automata rather than purely discrete models. We use automata rather than programming language models[Cousot 1977].

CHARON and Hybrid I/O Automata (HIOA) exhibit many of these concepts[Alur et. al. 2001, Lynch et. al. 2003]. The notion of abstraction used in this paper also involves containment of reachable states or traces. We allow

looser definitions than the CHARON notion of refinement or the HIOA notion of implementation, for example we allow the sets of abstract and concrete variables to differ. We allow fairly arbitrary abstractions to be specified and focus on verifying that they are adequate for the purpose at hand. CHARON and HIOA use more traditional ways to compose automata based on shared variables and/or shared events, whereas we use a scheduler function to compose models of real-time tasks that interact by contending for shared processors.

Markov (and more general stochastic) processes are well known to exhibit the state space explosion when trying to solve large models of complex systems. This served to motivate the desire to use more computationally tractable abstractions. Early work established necessary and sufficient conditions for when abstractions of Markov chains were again Markov [Kemeny and Snell 1976]. Considerable effort has been spent in developing efficient algorithms to find tractable Markov abstractions (*e.g.* [Derisavi et al. 2003a]). Other researchers have sought abstractions for which the solution is exact when the concrete model is a semi-Markov processes, which is more expressive than a Markov process[Bradley et al. 2003]. When a Markov process has no tractable abstraction that is again Markov, techniques for finding approximate abstractions might be useful [Lefebvre 2002].

From a computer science perspective, process specifications typically begin with models of concurrent automata, to which various stochastic semantics have been applied. Considerable work has gone into linking conditions for when variants of stochastic automata are analyzable as Markov chains (*e.g* [Brinksma and Hermanns 2001, Desharnais et al. 2003]). Software tools have been developed to support specification of numerous modeling formalisms and abstractions coupled with a collection of optimized solution techniques for evaluating them (*e.g.* [Derisavi et al. 2003b]).

3. Timing Models

Classical real-time scheduling theory deals with the scheduling and analysis of repetitively dispatched tasks[Liu and Deitel 2000]. The time between dispatches is fixed (periodic tasks) or has a lower bound (sporadic tasks). There is an upper bound on the compute time at each dispatch (often called the worst-case execution time). The theory provides algorithms for optimal (in some sense) uni-processor scheduling and for tractable schedulability analysis of large sets of tasks. However, classical real-time scheduling theory deals with only very restricted forms of internal task behaviors or interactions between tasks (beyond contention for a shared processor resource). For example, tasks in an actual system may exist in a number of discrete states, e.g. halted, initializing, suspended, computing, recovering.

Hybrid automata can model more complex dynamical systems[Alur et. al. 1994]. A hybrid automaton is a classical finite state automaton plus a set of

real-valued variables. The variable values may change continuously in a fixed location (a fixed discrete state), and may change discontinuously (may be assigned) at discrete transitions between locations. The allowed transitions may depend on the variable values (edge guards may be predicates over variables). These additional behaviors are specified by annotating the edges and locations of the classical finite state automaton with various kinds of constraints. In this paper we limit our attention to linear hybrid automata, where constraints are expressed using linear functions. A state of a hybrid automaton consists of a location together with a real value for each variable. We use *polyhedron* to refer to a set of possible real values for the variables (e.g. specified as a system of linear inequalities), and use *region* to refer to a location plus a polyhedron. Composition rules exist to define semantics for sets of concurrent hybrid automata.

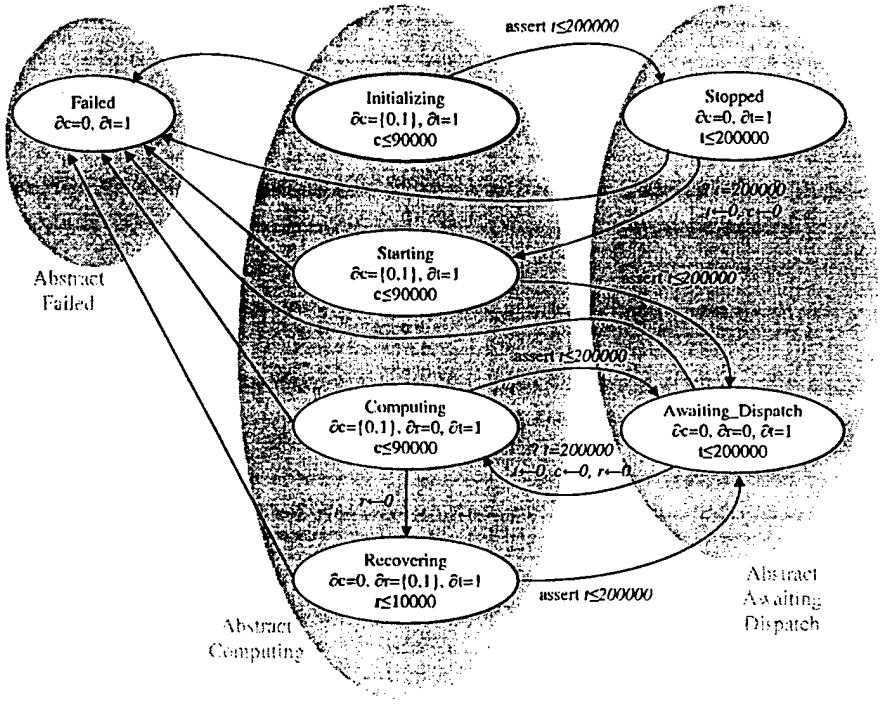

Figure 1. Concrete Hybrid Automata Model T for a MetaH Periodic Task

Certain AADL thread semantics are defined in the standard using a hybrid automata notation[AADL 2004]. We have automatically generated linear hybrid automata models for the portions of the MetaH middleware that perform preemptive scheduling and enforce time partitioning[Vestal 2000]. Figure 1

shows a hybrid automata model T for a periodic task. This model was automatically generated from the MetaH middleware code, i.e. it shows task behavior actually implemented by the middleware (excluding stopping and restarting at dynamic architecture reconfigurations). We use δx as an abbreviation for $\delta x/\delta t$. The choice for $\delta c = \{0, 1\}$ is made as follows.

We do not use shared variables or shared edge labels (synchronized transitions) to compose multiple automata. Instead, we use a scheduling function that defines the rates at which compute times accumulate as a function of the current set of task locations (e.g. as a function of which tasks are in ready states)[Vestal 2000]. Let $\bar{l} =< l_{1i}, l_{2j}, ... >$ be a location vector for a system of automata, i.e. l_{1i} is a location from automaton T_1, l_{2j} is a location from automaton T_2, etc. A scheduler function $< \delta v_1, \delta v_2, ... >= S(< l_{1i}, l_{2j}, ... >)$ (also written $\delta \bar{v} = S(\bar{l})$) defines the variable rate vector as a function of the system location vector. In our example, the scheduler function always sets $\delta t = 1$ for timers t, and sets $\delta c_i = 1$ if task i is executing and $\delta c_i = 0$ if task i is preempted for that system location (for that set of contending ready tasks).

Unfortunately, analyzing schedulability by model-checking systems of hybrid automata is not currently very tractable. We have done this for pairs of different kinds of tasks during the MetaH middleware verification exercise, but revolutionary advances in hybrid automata model-checking are needed to consistently analyze even a dozen non-trivial concurrent task models. We instead explore how to verify that a complex hybrid automaton task model (such as one defined in the AADL standard) can be safely approximated by a classical real-time task model for the purpose of schedulability analysis.

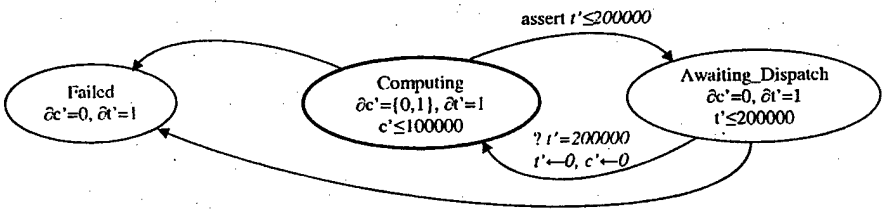

Figure 2. Abstract Hybrid Automata Model T' for a MetaH Periodic Task

Figure 2 shows an abstract hybrid automaton specification T' for a periodic task having a period of 200000 time units and a worst-case compute time of 100000 units. We assert this formally specifies a classical periodic real-time task, slightly extended by the addition of a Failed state. We define a mapping between this abstract automaton and the concrete automaton of Figure 1 as follows.

We define a many-to-one mapping of concrete to abstract locations, $l' = a(l)$ for abstract location l' and concrete location l. Every initial concrete lo-

cation must map to an initial abstract location. Our example mapping is illustrated in Figure 1 using shaded ovals to represent the abstract locations to which the concrete locations are mapped. We define the value of each abstract variable as a linear function of the concrete variables, $v'_i = f_i(v_1, v_2, ...)$ for each abstract variable v'_i and concrete variables v_j (also written $\bar{v}' = \bar{f}(\bar{v})$) . For our example, $t' = t$ and $c' = c + r$. Each initial valuation for the concrete variables must map to an initial valuation for the abstract variables.

Assume we are given a system of abstract tasks $T'_1, ..., T'_i, ...$ having an abstract scheduler function $\delta\bar{v}' = S'(\bar{l}')$. We can view this as an abstract specification for scheduling a system of tasks. We can modify this system by replacing some particular T'_i with a concrete T_i, with suitable changes to the domain and range of the scheduler function.

We constrain the modified scheduler function S obtained from the abstract S' so that all concrete locations that map to the same abstract location are equivalently scheduled, and concrete scheduler rates are consistent with abstract scheduler rates. Assume that, due to the replacement of T'_i by T_i, abstract variable v'_i is removed from the range of S and concrete variables $v_{i1}, ...$ where $v'_i = f_i(v_{i1}, ...)$ are added. For unreplaced abstract variables v'_j, $\delta v'_j = S'_j(< ..., l', ... >) = S_j(< ..., l, ... >)$ whenever $a(l) = l'$. For substituted variables, $\delta v'_i = S'_i(< ..., l', ... >) = \delta f_i(v_{i1}, ...)$ with $\delta v_{ij} = S_{ij}(< ..., l, ... >)$ whenever $a(l) = l'$.

We assert that the original abstract system can be analyzed using a classical schedulability analysis algorithm appropriate to the abstract scheduling function S'. If the reachable regions of the modified system are contained in those of the original abstract system (after applying the variable abstraction function) for all feasibly scheduled abstract systems, we assert that the abstract system is a safe approximation for the modified system for the purpose of schedulability analysis.

To formalize the notion of containment in the presence of variable abstraction, let $P'^{\downarrow}$ be the system of linear inequalities obtained from an abstract P' by substituting for each abstract variable v'_i its linear abstraction function $f_i(v_1, v_2, ...)$. Only concrete variables appear in $P'^{\downarrow}$. We say that concrete P is contained in abstract P' if $P \subseteq P'^{\downarrow}$

We verify by model-checking that a modified S derived as explained above from a feasible abstract scheduler S' will always feasibly schedule T_i. First, for our example pair of abstract and concrete models we restrict our attention to schedulers that are functionally equivalent to the set of constant rate schedulers $S'(..., \texttt{Computing}'_i, ...) \geq 1/2$, i.e. an abstract scheduling function is feasible for this example if it allocates at least 50% of the processor to T'_i between its release time and deadline while T'_i is in its compute state. Second, we construct a specific S that satisfies the conditions above, one that sets $\delta c = 1/2$ and $\delta r = 0$ in all concrete states that map to the abstract computing, except $\delta r = 1/2$

and $\delta c = 0$ in the recovering state. For our MetaH example, both abstract and concrete scheduler functions are preemptive fixed priority schedulers. (Note that, as one might expect, a number of concrete schedulers could be defined that satisfy the above conditions on the relation between abstract and concrete scheduler functions.)

Using these abstract and concrete scheduler functions, we applied a region enumeration tool to both an abstract and a concrete task model. Then, for each reachable concrete region (l, P) where l is a concrete location and P a polyhedron in the concrete variable space, the tool verified that there was some reachable abstract region (l', P') such that $l' = a(l)$ and $P \subseteq P'^{\downarrow}$. Note this is a conservative containment test, sufficient but not necessary, because in principle P might be contained in a union of abstract polyhedra but not in any single abstract polyhedron.

The condition that S is indistinguishable from S' for all concrete locations that map to the same abstract location means the scheduling of a given task model is the same regardless of whether it is being composed with abstract or with concrete models. We can thus make this substitution for any arbitrary subset of tasks to produce mixed-fidelity models that range from all abstract to all concrete.

This worked for our example concrete MetaH task model by design, in the sense that the task scheduling implementation was designed to present a classical real-time workload. This enabled accurate schedulability analysis for implemented systems, at least to the degree we could verify the implementation satisfied the abstraction (subsequent hybrid system model generation and checking revealed some implementation defects[Vestal 2000]). The advent of hybrid automata methods (largely occuring after the original MetaH design) and abstraction methods (such as those presented here) can hopefully enable more rigorous and defect-free development in the future.

Abstraction methods such as that presented here might be used to produce mixed-fidelity hybrid automata models that are more tractable to model-check. Our earlier experience suggests that expanding only two or three out of a dozen abstract tasks into their fully detailed concrete models might yield a tractably analyzeable model[Vestal 2000]. This might be useful, for example, to verify some complex interaction protocol between a pair of tasks.

Our use of model-checking to verify containment of concrete behavior within abstract behavior required us to constrain the class of abstract and concrete schedulers and the mapping between them. It would be useful to verify that the abstraction is a safe approximation for the concrete for broad classes of abstract and concrete schedulers and mappings. For example, it might be possible to permit a (mapped) concrete scheduler rate to exceed the abstract rate under certain circumstances. This might make it easier to deal with things like different scheduling priorities for different concrete locations, or bounded blocking

times, which would be of significant practical utility. It might also be possible to prove more complex cases of containment using an explicit detailed abstraction mapping between concrete and abstract invariants and edges (including guards and assignments), rather than model-checking with constrained scheduler functions.

4. Safety Models

We now revisit the same general problem addressed in the previous section, but rather for safety models than for timing models. The AADL Error Modeling Annex defines language features to specify stochastic models for fault, error and failure behaviors in embedded computer architectures[AADL 2004]. A stochastic automaton approach is used[Brinksma and Hermanns 2001] for specification. The rules for composing individual component stochastic automata depend on the specified architectural structure, i.e. depend on the possible error propagation paths between components that interface to or depend on each other. Propagation modifiers can be specified to make propagation conditional, which allows consensus and voting protocols to be modeled.

An error model for a system specified as a nested hierarchy of components can be obtained by composing the error models for its subcomponents according to the rules of the language. However, another option is made available: the user can specify a subsystem error model that may optionally be substituted as an abstraction for the concrete compositional model. Propagation modifiers can also map one error into another. This makes it easier to compose legacy models or models developed at different levels of abstraction. (Legality rules are included in the annex to enable automatic verification of error model compatibility within an overall architecture specification, or identify places where such mappings are needed.)

The remainder of this section is organized as follows. We introduce Markov processes, the modeling language to which stochastic automata specifications are translated before solving the system. We suggest rules to preserve safety properties when going from concrete (larger) steady state Markov models to abstract (smaller) steady state stochastic models. Abstractions of several steady state Markov models are presented. Steady state analyses are computationally much simpler to find than transient analyses.

We show that the transition rate assignment in the abstract model is uniquely determined by the transition rates of the concrete model when the abstraction is "lumpable". When the abstraction is not lumpable, rate assignments in an abstract model need not be uniquely determined. We discuss selection criteria for "reasonable" assignments from an engineering perspective when possibly infinite (beyond a constant rescaling of all transition rates) assignments will satisfy the constraints of the abstract model. For safety analyses, transient solutions are generally required. We discuss conditions for preserving safety

in transient models. We close with an illustration of how Markov chains are composed at the (AADL) specification level.

4.1 Brief Markov Process Introduction

The reader is assumed to be familiar with Continuous Time Markov Chains (CTMCs) at an introductory text level (e.g. [Hoel et. al. 1972]). We use standard notation for describing CTMCs, which unfortunately has some overlap with hybrid systems notation. Hopefully the context will make clear the use. The notation we use to specify and solve CTMCs is compactly defined in Table 1. When considering limiting distributions, we assume there are no death states and the limiting distribution does not depend on the initial distribution. That is, we assume the CTMC is ergodic and regular.

Notation	Description
δ	A finite discrete set of systen states. Typically, $\delta = \{1, 2, ..., m\}$.
x, y	Elements in δ. $x, y \in \delta$.
$X(t)$	System state at time $t \geq 0$. $X(t) \in \delta$ for all $t \geq 0$.
q_{xy}	Instantaneous rate of change from state x to y for $x \neq y$. The set $\{q_{xy}\}$ describes the infinitesimal generators of the CTMC. For $x = y$, $q_{xx} = -q_x = -\sum_{y \in \delta - \{x\}} q_{xy}$. In practice, q_{xy} is known or must be approximated (*e.g.* the failure rate of a component, perhaps given by a vendor specification).
$\mathbf{A}$	The infinitesimal generator matrix. Denote $(\mathbf{A})_{ij} = q_{ij}$.
q_x	The transition rate out of state x. For a CTMC, this means the probability that a process in state x will remain in state x for a time greater than t is $e^{-q_x t}$. If x is a death state (with no transitions leaving x), then $q_x = 0$.
D	A diagonal matrix, with $D_{xx} = q_x$ and $D_{xy} = 0$ for $x \neq y$.
Q_{xy}	The probability of transition from state x directly to state y given the system is about to transition out of $x \neq y$. $Q_{xy} = q_{xy}/q_x$ for $x \neq y$.
$P_{xy}(t)$	The probability that $X(t) = y$ given that $X(0) = x$. Or, the probability that a process X in state x will be in state y after t time has elapsed.
π	The steady state distribution. That is $\pi = (\pi_1, \pi_2, ..., \pi_m)$, where $\pi_x = \lim_{t \to \infty} P(X(t) = x)$. For a CTMC, π satisfies $\pi\mathbf{A} = 0$. For a Discrete Time MC (DTMC), π satisfies $\pi Q = \pi$. Also require $\sum_{j=1}^{m} \pi_j = 1$, to fully constrain the model.

Table 1. Continuous Time Markov Chain (CTMC) Notation

4.2 Examples of Concrete Continuous Time Markov Chain Models

We give three Markov models used in subsequent examples. Models are concrete when no further detail is captured in any of the states or transitions.

The model shown in Figure 3 is the simplest possible Markov process that can represent a single repairable component (SRC). The right hand side shows standard notation. The left hand side is an equivalent, yet more compact representation that we adopt. In Figure 3, $\delta = \{1, 2\}$. When in the operational state (1), faults occur at rate λ, when the process transitions to the failed state (2). The failed system returns to operational when the repair event has been

effected, which occurs at rate μ. When repairs are not instantaneous, the repair completion time is equated with the repair event epoch. Table 2 summarizes

Figure 3. Failure/Repair Transition Notation and SRC Model

these transitions and gives the steady state distribution.

$x \in \delta$	(x, y)	q_{xx}	q_{xy}	π_x
1	$(1, 2)$	$-\lambda$	λ	$\mu \cdot (\mu + \lambda)^{-1}$
2	$(2, 1)$	$-\mu$	μ	$\lambda \cdot (\mu + \lambda)^{-1}$

Table 2. Single Repairable Component Markov Process Specification

For our second example, we consider an abstraction that aggregates a sequence of events, which may be desirable in practice. Figure 4 show a process consisting of a sequence of four events reduced to three events.

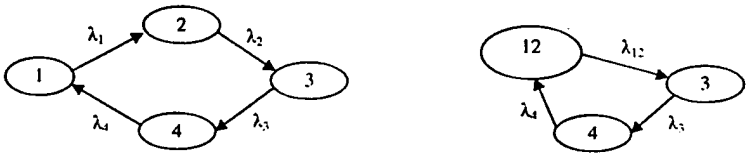

Figure 4. Markov Cycle Models (Right abstracts Left)

The last example is a triple modular redundancy (TMR) system with three independent and identical components, C_1, C_2, and C_3. Components are either working or failed, with failure and repair rates λ and μ, respectively. System state is defined by the state of all components, with "operational" states as two or more components are working. Figure 5 and Table 3 show the TMR Markov process, parameters, and steady state solution.

4.3 Safe Abstractions of Concrete Models

Superscripts a and c are used to distinguish between abstract and concrete models. For example δ^a and δ^c denote abstract and concrete states, respectively. To ensure safety properties, we propose two rules for defining abstract models in terms of concrete models.

(1) To ensure that concrete states are not split and distributed among multiple abstract states, we recommend that the concrete states are partitioned where

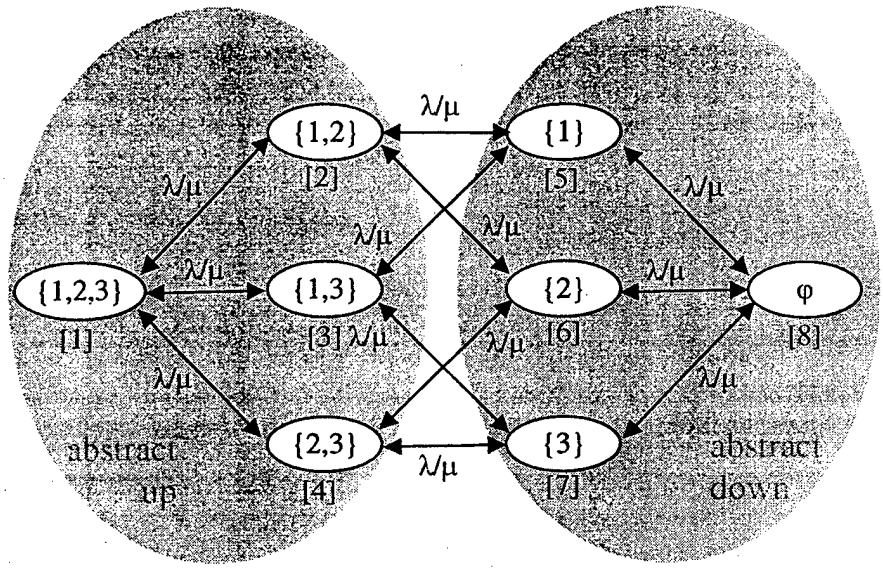

Figure 5. Markov TMR Model

op comps	state x	q_x	up?	π_x	Abs 1	Abs 2
$\{1,2,3\}$	1	-3λ	yes	$\mu^3 \cdot (\mu+\lambda)^{-3}$	P_1^{a1}	P_1^{a2}
$\{1,2\}$	2	$-(2\lambda+\mu)$	yes	$(\mu^2\lambda) \cdot (\mu+\lambda)^{-3}$	P_2^{a1}	
$\{2,3\}$	3	$-(2\lambda+\mu)$	yes	$(\mu^2\lambda) \cdot (\mu+\lambda)^{-3}$		
$\{1,3\}$	4	$-(2\lambda+\mu)$	yes	$(\mu^2\lambda) \cdot (\mu+\lambda)^{-3}$		
$\{1\}$	5	$-(\lambda+2\mu)$	no	$(\mu\lambda^2) \cdot (\mu+\lambda)^{-3}$	P_3^{a1}	P_2^{a2}
$\{2\}$	6	$-(\lambda+2\mu)$	no	$(\mu\lambda^2) \cdot (\mu+\lambda)^{-3}$		
$\{3\}$	7	$-(\lambda+2\mu)$	no	$(\mu\lambda^2) \cdot (\mu+\lambda)^{-3}$		
$\emptyset$	8	-3μ	no	$\lambda^3 \cdot (\mu+\lambda)^{-3}$	P_4^{a1}	

Table 3. TMR Markov process specification for Figure 5

each partition corresponds to a single abstract state. When $\delta^a = \{1^a, 2^a, ..., m^a\}$ then a partition on $\delta^c = \bigcup_{i=1}^{m^a} P_i^c$ is defined so that $j^a \equiv \{x | x \in P_j^c\}$ and $j^a \cap i^a = \emptyset$ for $j^a \neq i^a$. For a "safe" steady state abstraction, assign probabilities to the abstract states by:

$$\text{For } x \in \delta^a, \text{ assign } \pi_x^a = \sum_j \pi_j^c, \text{ where } j \in P_x^c \cap \delta^c.$$

When error states are aggregated in an abstraction, this assignment ensures that the probability of the error in the abstraction is not reduced.

This state aggregation (or partitioning) rule is consistent with the abstraction model of heirarchical decompositions. It is also intuitive when system states correspond to the (discrete) operational condition of physical components. For dependent faults an abstraction that "splits probabilities" across states might result in a better approximation. Further investigation is needed to determine if this heirarchical decomposition rule eliminates a number of useful abstractions.

(2) We further suggest that a one step transition from $x \in \delta^a$ to $y \in \delta^a$, $q^a_{xy} > 0$ only if there exists some $x' \in P^c_x \subset \delta^c$ and some $y' \in P^c_y \subset \delta^c$ such that $q^c_{x'y'} > 0$. This preserves a notional mapping from the abstract model to the system through the established mapping from the concrete model to the system. More importantly, it implies that errors in the abstract model cannot propagate in ways that were not specified in the concrete model.

4.4 Transition Rate Assignments for Safe Abstractions

We give three examples of safe steady state probability assignments for abstractions using the two step process in Section 4.3. We investigate the relationship between safe probabilities and rate assignments.

The right side of Figure 4 shows an abstraction of a four cycle model which merely collapses two states into one. Equation 1 gives the steady state solution of the concrete cyclic model in Figure 4.

$$\pi^c = (\pi^c_1, \pi^c_2, \pi^c_3, \pi^c_4) = \frac{(\lambda^c_2\lambda^c_3\lambda^c_4, \lambda^c_1\lambda^c_3\lambda^c_4, \lambda^c_1\lambda^c_2\lambda^c_4, \lambda^c_1\lambda^c_2\lambda^c_3)}{\lambda^c_2\lambda^c_3\lambda^c_4 + \lambda^c_1\lambda^c_3\lambda^c_4 + \lambda^c_1\lambda^c_2\lambda^c_4 + \lambda^c_1\lambda^c_2\lambda^c_3} \qquad (1)$$

For the reduced model on the right of Figure 4, a similar computation gives $\pi^a = (\pi^a_{12}, \pi^a_3, \pi^a_4)$ in terms of transition rates $\lambda^a_{12}, \lambda^a_3$ and λ^a_4. A solution that preserves exiting transition rates in non-aggregated states of $\mathbf{A}^a$ is

$$\lambda^a_{12} = (\lambda^c_1\lambda^c_2)(\lambda^c_1 + \lambda^c_2)^{-1}; \quad \lambda^a_3 = \lambda^c_3; \quad \text{and } \lambda^a_4 = \lambda^c_4. \qquad (2)$$

The solution in Equation 2 is not unique (four concrete parameters define three abstract parameters).

For the TMR example of Figure 5 we consider two abstractions. The two right most columns of Table 3 define the abstraction partitions. Abstraction 1, which defines abstract states by the number of operational components is shown in Figure 6. Section 4.5 shows this is a lumpable abstraction with unique

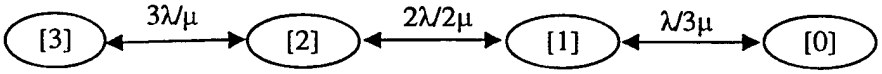

Figure 6. Number of Operational Components TMR Abstraction

(relative to the concrete model) transition rates, and how to compute them.

A courser abstraction of the TMR model is simply the two state model, $\delta^a = \{1, 2\} = \{\text{up}, \text{down}\}$. This abstraction is shown with shading in Figure 5 and also in the right most column of Table 3. When approximated by a

Markov process, this abstraction is represented in Figure 3. Equation 3 is the result of equating the two formulations for π^a, which does not have a unique assignment. The abstract model parameters must sastisfy

$$\lambda^a/\mu^a = (\lambda^c/\mu^c)^2 \cdot (\lambda^c + 3\mu^c)/(3\lambda^c + \mu^c) \tag{3}$$

In general, partitioned (abstract) processes are not Markovian, in which case the rate assignment need not be uniquely determined. The question is which assignment of values produces the best results from an engineering perspective. Is it preferable to hold constant the flow in, the flow out, the ratio of the flow in to the flow out, or some other property? One can envision practical circumstances which would favor each of these decisions.

4.5 Lumpability, Safe Abstractions and Rate Assignments

We define necessary and sufficient conditions for when the partitioned abstraction is again Markovian. Our discussion of strong lumpability for DTMCs follows the presentation in [Kemeny and Snell 1976].

Consider a partition P on δ with $k < m$ elements. Define $U_{k,m}$ and $V_{m,k}$ according to P as follows. The j^{th} row of U puts a probability distribution on the elements in P_j. For example, if P_j contains b_j states over which the uniform distribution is to be placed, then

$$U_{j,s} = \begin{cases} 1/b_j & \text{for } s \in P_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The rows of a matrix V define the partition to which the state belongs. *I.e.*

$$V_{s,j} = \begin{cases} 1 & \text{for } s \in P_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Theorem 1 gives conditions for strong lumpability with respect to partition P of a Discrete Time Markov Chain (DTMC).

THEOREM 1 (DTMC STRONG LUMPABILITY) *Let P be a partition for the DTMC with state space δ and transition matrix Q. Let U and V be matrices defined by Equations 4 and 5 with respect to P. The DTMC is said to be strongly lumpable with respect to P if and only if*

$$VUQV = QV.$$

For a proof, see Theorems 6.3.4 and 6.3.5 of [Kemeny and Snell 1976].

Theorem 2 is an easily obtained analog for conditions of strong lumpability in a Continuous Time Markov Chain (CTMC).

THEOREM 2 (CTMC STRONG LUMPABILITY) *Let P be a partition for the CTMC with finite state space δ and infinitesimal generator matrix A. Let U and V be matrices defined by Equations 4 and 5 with respect to P. The CTMC is said to be strongly lumpable with respect to P if*

$$VUD^{-1}AV = D^{-1}AV$$

where $D = -diag(A)$. *That is,* D *is a diagonal matrix with* $(D)_{ii} = -(A)_{ii}$.

To show this result, note that the DTMC transition matrix $Q = D^{-1}A + I$. *An application of Theorem 1 gives*

$$VU(D^{-1}A + I)V = (D^{-1}A + I)V.$$

Since $UV = I$, *the result follows.*

The rates for the abstract model are found by computing $A^a = UA^cV$. An algorithm for finding the coursest (*i.e.* the most abstract) strongly lumpable model is given in [Derisavi et al. 2003a]. This algorithm has computational complexity $O(|Q^c| \cdot \log_2(|\delta^c|))$ and space $O(|Q^c| + |\delta^c|)$, where $|Q^c|$ is the number of positive transitions in the concrete model.

Weak lumpability occurs when the lumped process is Markov when starting from some (but not all) initial distributions ([Kemeny and Snell 1976]). Work has been done linking both strong and weak lumpability MP results to the same properties in stochastic automata(*e.g.* [Brinksma and Hermanns 2001]).

Investigation as to whether lumpable partitions create natural and useful abstractions for system models is needed. When an abstraction is not lumpable, a measure of "near lumpability" has been proposed as a measure of the quality of the approximation.

4.6 Time Dependent or Transient Solutions

For a time dependent analysis, we define safety for an abstract model with partition P as follows. Let $x \in P_s \subset \delta^c$ be a non-fault or safe set of states and $y \in P_f \subset \delta^c$ be a "fault occurence" set of states. The abstraction is said to be safe in the time interval $[0, T]$

$$P_{s^a}(X^a(t) = f^a) \geq P_x(X(t) \in P_f) \; \forall \, x \in P_s \text{ and } \forall \, t \in [0, T]. \qquad (6)$$

In words, we require for all $t \in [0, T]$ that when starting in safe abstract state s^a, the probability of reaching abstract fault state f^a is at least as great as the probability of reaching any state in partition P_f when starting from in any state in partition P_s in the concrete model.

When the concrete Markov process is started in steady state π^c, then for every time $t > 0$ and for all $x \in \delta^c$, the $P_\pi(X(t) = x) = \pi_x$. When the abstraction is strongly lumpable (hence Markovian), the requirements of Equation 6 are satisfied because probabilities sum within partitions and the distribution of time to transition from all states in a partition to another partition is the same.

We are not sufficiently familiar with the literature to be able to report whether a transition assignment that can satisfy the requirements of Equation 6 exists for an arbitrary complex fault model with a non-lumpable abstraction. Perhaps an equally important question is how those conditions might be applicable for guiding the development of practical fault models.

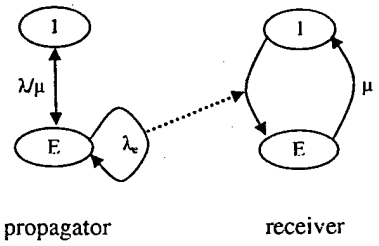

Figure 7. Error Propagation Between Markov Models

4.7 Composing Concurrent Models

Figure 7 illustrates the basic idea behind composing multiple Markov chain component models, one Markov Chain per component. The user may distinguish selected states as error propagating states, which is modeled as a self-transition with a given error propagation rate. For an error that may propagate from one component to another (determined by the architecture specification), the rate of a transition in the receiving model is determined by the rate of the propagating transition rather than a rate specified in the receiving model. A fundamental result of stochastic process algebras is that, under suitable restrictions, such rendezvous between concurrent stochastic processes have Poisson rates. Once this rate has been determined it can be used for the rate within the receiving model, and the methods of the preceeding sections applied to verify an abstraction. Similarly, self-transitions can be added to an abstract model to define propagation rates to be used in other receiving models.

The AADL Error Model Annex includes a way to define guards on error transitions to model things like voting and consensus protocols. In other words, additional language features and semantics are included to compactly specify complex event propagation conditions. More research is needed to determine when high level abstractions are closely approximated by the generated Markov abstractions.

5. Future Work

We have given only two examples of techniques that can be used to demonstrate that an abstract model can safetly (in some sense) be substituted for a more complex concrete model when generating hybrid and stochastic automata models from architecture specifications. Preliminary approaches for linking MetaH/AADL safety specifications with concrete and abstract Markov models with solvers have been reported [Binns et al. 2000]. A more complete toolbox is needed. Also, more complex notions of abstraction may be useful, for example conformance relations[Krichen and Tripakis 2004].

References

[AADL 2004] SAE AS5506, *Architecture Analysis and Design Language*, Society of Automotive Engineers, Warrendale, PA, 2004.

[MetaH 2000] *MetaH User's Guide*, Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN, `www.htc.honeywell.com/metah`.

[Alur et. al. 2001] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, R. Mishra, G. Pappas and O. Sokolsky, "Hierarchical Hybrid Modeling of Embedded Systems," EMSOFT 2001, Springer Verlag LNCS 2211, 2001.

[Alur et. al. 1994] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, R.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *International Conference on Analysis and Optimization of Discrete Event Systems*, LNCIS 199, Springer-Verlag, 1994.

[Binns et al. 2000] P. Binns, S. Vestal, W. Sanders, J. Doyle, and D. Deavours, "MetaH/Mobius Integration Report", Customer Report for DARPA's Evolutionary Design of Complex Systems (EDCS) Program, Honeywell Labs, April 2000.

[Bradley et al. 2003] Jeremy Bradley, Nicholas Dingle, and William Knottenbelt, *International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (SPECTS), July 2003

[Brinksma and Hermanns 2001] Ed Brinksma and Holger Hermanns, "Process Algebra and Markov Chains," Springer LNCS 2090, *European Educational Forum: School on Formal Methods and Performance Analysis*, 2001.

[Cousot 1977] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," Sixth Annual Symposium on Principles of Programming Languages, Los Angeles, California, 1977.

[Derisavi et al. 2003a] S. Derisavi, H. Hermanns, and W. H. Sanders, "Optimal State-Space Lumping in Markov Chains," *Information Processing Letters*, vol. 87, no. 6, September 30, 2003,

[Derisavi et al. 2003b] S. Derisavi, P. Kemper, W. Sanders, and T. Courtney, *Performance Evaluation*, Volume 54(2), October 2003

[Desharnais et al. 2003] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panagaden, "Metrics for Labelled Markov Processes," to appear *Theoretical Computer Science*, Elsevier

[Hoel et. al. 1972] Paul G. Hoel, Sidney C. Port, and Charles J. Stone, *Introduction to Stochastic Processes*, Houghton Mifflin Company, USA, 1972.

[Kemeny and Snell 1976] John G. Kemeny and J. Laurie Snell, *Finite Markov Chains*, Springer-Verlag, 1976.

[Krichen and Tripakis 2004] Moez Krichen and Stavros Tripakis, "Black-box Conformance Testing for Real-Time Systems," SPIN'04 Workshop on Model Checking Software, LNCS 2989, 2004.

[Lefebvre 2002] Yannick Lefebvre, "Approximate aggregation and applications to reliability," *Third International Conference on Mathematical Methods on Reliability (MMR)*, 2002

[Liu and Deitel 2000] J. Liu and P. Deitel, *Real-Time Systems*, Prentice-Hall, New Jersey, 2000

[Lynch et. al. 2003] Nancy Lynch, Roberto Segala and Frits Vaandrager, "Hybrid I/O Automata," *Technical Report MIT-LCS-TR-827d*, MIT Laboratory for Computer Science, Cambridge, MA, Jan. 13, 2003; and *Information and Computation*, 185(1). Aug. 2003

[Milner 1989] Robin Milner, *Communication and Concurrency*, Prentice Hall, UK, 1989

[Vestal 2000a] Steve Vestal, "Formal Verification of the MetaH Executive Using Linear Hybrid Automata," *Real-Time Applications Symposium*, June 2000.

[Vestal 2000] Steve Vestal, "Modeling and Verification of Real-Time Software Using Extended Linear Hybrid Automata," *NASA Langley Formal Methods Workshop*, June 2000.

Real-Time Sampled Signal Flows through Asynchronous Distributed Systems

Steve Vestal

Abstract- **We present a new model of real-time periodic distributed asynchronous computation in which information flows through sequences of periodic tasks, where task inputs are obtained by sampling other task's outputs as well as sampling the environment. We introduce a metric for end-to-end timing called the age of an output, which is the time since the external inputs on which an output value is based were sampled. We present some bounds on age scheduling efficiency, reduce the problem of finding a feasible distributed age schedule to finding a solution for a system of nonlinear constraints, and discuss use of a commercial solver to find a solution to a large problem derived from a real-world system.**

Index Terms- real-time, asynchronous, distributed systems

I. INTRODUCTION

Many large distributed control systems are built by plugging together components that operate periodically using their own internal clocks. For example, a sensor periodically samples the environment, a bus periodically conveys data from the sensor to a processor, the processor periodically executes tasks that operate on sensor data. If all these components operate using separate and unsynchronized clocks, then we could say that the bus periodically samples the sensor and the processor periodically samples the bus, analogous to the way the sensor periodically samples the environment. We present a model for this kind of distributed asynchronous system, one in which sampling may occur at internal asynchronous interfaces between components.

Perhaps the most widely-used metric for end-to-end timing in distributed system is latency, which is the time between the arrival of an external data value and the time at which the corresponding output value arrives at its final destination. However, this metric does not work well for our model for at least three reasons. First, while it is reasonably clear what latency means for a linear sequence of precedence-constrained tasks, it is not as clear what latency means for less restricted connectivity graphs in which an output may be computed from multiple inputs. Second, this metric applies to each data value, which passes in a loss-less manner from task to task. In our model, values may be lost internally, for example due to under-sampling of a data flow that goes from a high-rate to a low-rate task. Third, it is not additive in the sense that end-to-end latency is not the sum of the latencies through each task. We present a new metric called the age of an output value. Intuitively, the (worst-case) age of an output value is the (worst-case) time since the external input values on which that output is based were sampled.

An age scheduler selects task sampling periods as well as decides which among a set of ready tasks is executing on each processor. For uni-processor age scheduling, we derive bounds on achievable age utilization, and on the ratio of classical to aged utilization. For multi-processor age scheduling, we formulate the problem of selecting a set of good periods for each processor as a non-linear constraint satisfaction problem. We also bound the efficiency of an asynchronous sampling system relative to an idealized fast uni-processor. Finally, we describe the use of a commercial non-linear optimization tool to solve a problem derived from a large real-world avionics system.

II. PREVIOUS WORK

Most work on end-to-end timing in real-time systems models a task as a linear sequence of subtasks [1,3,4]. The first subtask in a sequence is released according to some temporal constraint (e.g. periodic, minimum inter-arrival time, arrival curve). Each subsequent subtask in the sequence is released when its predecessor finishes. The end-to-end latency for each task is defined to be the time between the release of the first subtask and the completion of the last subtask in that sequence. The worst-case end-to-end latency for that task is the largest such value for all releases of that task.

We present a different model for distributed asynchronous systems that, to our knowledge, does not appear in the literature. We restrict our attention to tasks having periodic release times. In our model, every subtask has its own periodic release, each subtask samples its predecessors. Each subtask is its own independent sampled data system, sampling all its inputs and using a zero-order hold for all its outputs. In the standard model, there is no loss of data between subtasks in a fault-free system. In this model, data may be lost due to under-sampling by one subtask of its predecessor subtask's outputs. In this model, we are not restricted to linear sequences of subtasks, the flow of data between subtasks can

This work was supported by Honeywell and the US Air Force Office of Scientific Research under contract F49620-01-C-0024. Steve Vestal is with Honeywell Laboratories, Minneapolis, MN 55418 USA (e-mail: Steve.Vestal@ Honeywell.com).

be an arbitrary directed graph.

Both the standard and our model are similar in the assumption that subtasks have a fixed allocation to processors. The allocation problem, and the scheduling and schedulability analysis problem, are handled separately.

III. ASYNCHRONOUS SAMPLING SYSTEMS

We first introduce our model and terminology for a periodic task hosted on a single processor, illustrated in Figure 1. A processor is a piece of hardware able to execute periodic tasks. A task hosted on a processor is periodically dispatched, where the dispatches are exactly T time units apart. A task can access input and output ports associated with its hosting processor. Following each dispatch, the processor will sample input ports and perform work for that task, scheduled in a manner chosen for the processor. When C units of work are completed following a dispatch, the task writes output values into output ports. The time between the dispatch and the writing of the output value is called the latency L. Note that latency is defined relative to the dispatch instant, not the times at which inputs are sampled.

We use processors and tasks to abstractly model several kinds of hardware resources and activities. Thus, a processor may model a computer, or a switch, or a bus or network link. A task may model a software computation or a message transmission.

We assume that T is the same between every dispatch, but that C and L may vary. Note that our definition allows inputs to be sampled and outputs to be written at different offsets from the dispatch for different dispatches (i.e. our model admits jitter). Unless specifically stated, we use C and L to denote the worst-case (supremum) work and latency for all dispatches, where L denotes the worst-case latency for any output of a task. We assume $0 < C \le T$ and $C \le L$, but not necessarily $L \le T$.

We think of reading an input port as sampling a continuous input signal, and writing an output port as setting the value of a continuous zero-order hold output signal. The intervals of time between successive samplings of an input and between successive settings of an output, although T on average, may vary somewhat due to the way a processor is scheduled.

A system has a set $\Theta = \{\rho_1, \rho_2, ...\rho_{N_\Theta}\}$ of N_Θ processors and a set $\Psi = \{\tau_1, \tau_2, ...\tau_{N_\Psi}\}$ of N_Ψ tasks[1]. We use Ψ_ρ to denote the subset of tasks hosted by processor ρ, where we assume a static assignment or binding of tasks to processors in this paper.

A system, illustrated in Figure 2, has a set $\Phi = \{\phi_1, \phi_2, ...\phi_{N_\Phi}\}$ of N_Φ flows. A flow ϕ is a cycle-free ordered sequence of tasks $\tau_{\phi 1}, \tau_{\phi 2}, ...\tau_{\phi N_\phi}$ of length N_ϕ. The first task in the sequence samples a designated input port, the last task in the sequence writes a designated output port, and each intermediate task samples an output of its predecessor and writes to an input of its successor. We use Ψ_ϕ to denote the subset of tasks in flow ϕ. Note that a task may appear in multiple flows. The set of flows is a specified subset of the paths in an arbitrary task graph between pairs of input and output ports (which need not be external ports).

We assume that every task is included in at least one specified system flow. However, we do not assume that a specification includes all possible flows in a system. Rather, we assume that the developer specifies a tractable set of flows that capture the timing requirements of the system.

[1] Throughout we use lower-case Greek letters to denote intuitively defined abstract entities such as processors and tasks, upper-case Greek letters to denote sets of such abstract entities, lower-case Roman letters to denote indices for elements of sets, and upper-case Roman letters to denote real-valued parameters of the models.

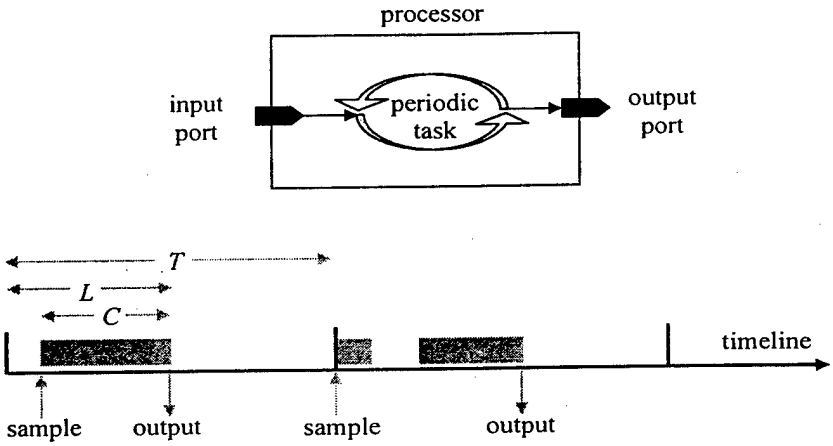

Fig. 1. Model and notation for a single periodic sampled-data task hosted on a single processor..

The interconnect topology between processor ports, the set of ports accessed by each task, and the hosting specifications Ψ_ρ, have some well-formed-ness constraints based on the specified flows. However, it is not necessary for the purposes of this paper to formalize these fairly intuitive constraints.

A fundamental assumption of our model is that processors operate asynchronously with respect to each other. Given any pair of tasks on different processors, we make no assumptions about the relative phase or offset of their dispatches, and our analyses cover all possible phasing. The scheduler on each processor may, however, control the phasings between the tasks hosted on that processor. As we will see, this can affect the end-to-end timing of flows.

IV. Flow Age

We now turn to the question of specifying system timing requirements. It turns out that defining a natural notion of end-to-end latency is not straight-forward. For example, the latency of the system is not the sum of the latencies along a flow. To see this, imagine a flow where all tasks have very small latencies but one of the tasks has a very large period. From a feed-back control perspective, both sampling period and latency are important, but we would like a metric that does not assign a special role to the period of the first (or last) task in a flow.

We introduce a new timing metric that we call the flow *age*. The age of a flow is the amount of time by which the output signal is out-of-date with respect to the input sample used to compute that output. The exact value of the age is a function of time. The age steps to some value when an output is written, then increases with time until the next output value is written. The worst-case age is the largest age that occurs at any point in time. We assume every output is initialized to some specified value at time 0, so that the initial age of an output equals the time since the system began operation until the first write is made to that output. We codify these concepts in the following results.

Lemma 1: The worst-case age of a flow that consists of a single task τ_i that samples an external input is $T_i + L_i$.

Proof: We first walk through the timeline of events assuming no variability in any of the parameters, illustrated in

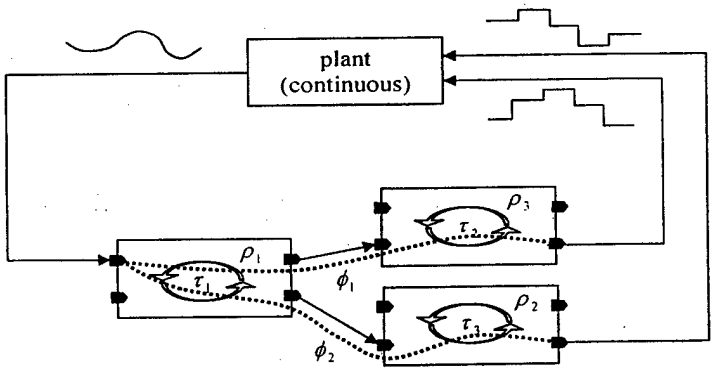

Fig. 2. Data flows through a system of asynchronous periodic sampling tasks and processors.

Figure 3. At the moment the input signal is sampled, the age of that sample value is zero. If this sampling occurs at the dispatch instant (the earliest time possible), then the age of the output signal becomes L_i at the moment the output value is written by the task. The age of this output continues to increase until the moment at which the next dispatch completes and writes a new output value (which has a smaller age). The time between successive completions of the task is T_i, so the worst-case age achieved of the output signal is $T_i + L_i$.

Looking at figure 3, if a sample is taken later than the dispatch instant, the effect is to shift the line showing the age for that sample to the right. This can only decrease the age of the output based on that sample. If the latency is less than the worst-case, so that an output is written earlier, this only moves a step-down of the dotted worst-case age function to the left. Thus, the result still holds in the face of allowed variability in sampling and completion times. §

Theorem 1: The worst-case age for a flow ϕ is the sum of the worst-case ages of each per-task flow, $\sum_{i \in \Psi_\phi} (T_i + L_i)$.

Proof: Assume the (worst-case) age of the input signal to task τ_i is A_{i-1} at the instant that input is sampled. At the instant the output value based on this sample is written, its (worst-case) age is $A_{i-1} + L_i$. The age of the output value continues to increase until it is overwritten by a subsequent value, which occurs T_i time units later. The result follows by observing that $a_0 = 0$ (the age of a value obtained by sampling the external input signal is always 0 at the instant of sampling) and induction on the number of tasks (every task just adds $T_i + L_i$ to the worst-case time elapsed since the original external input sample). §

It is fairly easy to see that the smallest possible age is $\sum_{i \in \Psi_\phi} L_i$.

Suppose the system is operating in such a way that the end-to-end age of a flow is arbitrarily close to this minimum, which will occur when the output of each task occurs arbitrarily close to the input of its successor. If the latency of all such tasks now increases very slightly, so the outputs occurs just after the inputs of the successor tasks, the flow suddenly goes from the least possible to the greatest possible end-to-end age. Such systems can in principle exhibit very high jitter and very rapid changes between small and large end-to-end age.

V. Uni-processor age scheduling

The traditional real-time periodic uni-processor scheduling problem has as inputs, for each periodic task τ_i, a specified period T_i; a specified upper bound C_i on the maximum compute time needed for any dispatch; and a specified upper bound on the allowed worst-case latency (often called the deadline D_i). The age scheduling problem uses the worst-case task age A_i as an input parameter, where the scheduler is free to trade-off period and latency as long as their sum remains less than the specified worst-case age.

We first explore the relationships of this problem to the traditional definition of uni-processor utilization,

$$U = \sum_{i \in \Psi_p} \frac{C_i}{T_i}$$

An important concept in traditional theory is that of the utilization bound for a particular scheduling algorithm. Given a particular scheduling algorithm S applied to a given class of real-time scheduling problems, the utilization bound U_S^* is that utilization below which a feasible schedule is guaranteed.

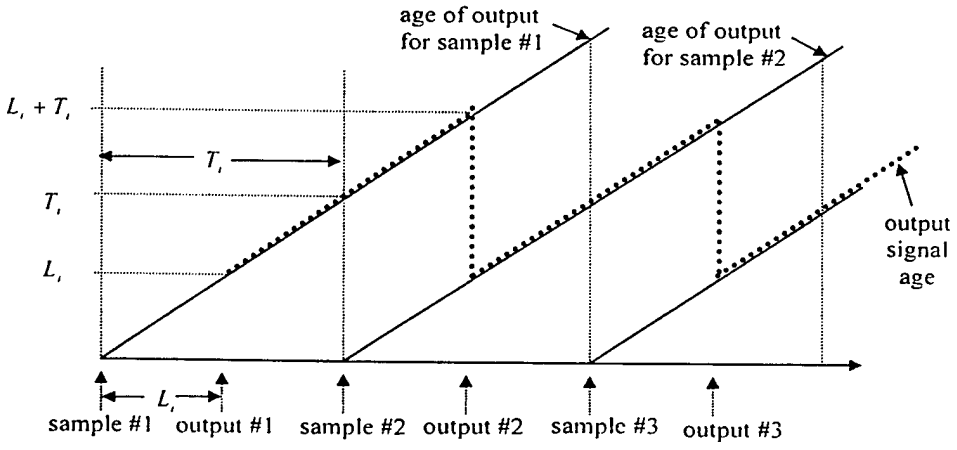

Fig. 3. Age of an output as a function of time.

That is, if $U \le U_S^*$ (where U is computed as above for the given problem) then that problem will be feasibly schedulable using algorithm S. The utilization bound can be used both to compare the efficiency of two scheduling algorithms, and as a simple test to see if a given problem can be feasibly scheduled by a given algorithm. However, we note that all known utilization bounds are inexact for arbitrary deadlines, in the sense there exist schedulable problems whose utilization is above the utilization bound. $U \le U_S^*$ is sufficient but not necessary to guarantee schedulability.

However, for age scheduling we cannot compute the traditional utilization from the problem statement, since the task periods T_i are determined by the scheduling algorithm. We thus explore some properties of an analogue value that we call the aged utilization,

$$U_A = \sum_{i \in \Psi_\rho} \frac{C_i}{A_i}$$

Lemma 2: Let U_S^* be the utilization bound for scheduling algorithm S for the traditional implicit deadline scheduling problem ($T_i = D_i$, the deadline for task completion is the next dispatch of that task). Then S with $T_i = A_i/2$ solves the age scheduling problem with an aged utilization bound of $U_S^*/2$ (i.e. any age scheduling problem where $U_A \le U_S^*/2$ will be feasibly scheduled by S with $T_i = A_i/2$).

Proof: The traditional implicit deadline requirement means $L_i \le T_i$, hence $T_i + L_i \le 2T_i$, hence it follows from $T_i = A_i/2$ that $T_i + L_i \le A_i$. Substituting $T_i = A_i/2$ into the formula that defines traditional utilization gives $U_A = U/2$, hence the aged utilization when the traditional utilization achieves U_S^* is $U_S^*/2$. §

Theorem 2: There exist algorithms to solve any age scheduling problem having an aged utilization no greater than 50%, and there exist age scheduling problems that cannot be feasibly scheduled by any algorithm when the aged utilization is greater than 50%.

Proof: To show the first part, we note that scheduling algorithms with a traditional utilization bound of 100% are known (e.g. earliest deadline, least laxity, harmonic rate monotonic). The preceding lemma shows that these can be applied to solve an age scheduling problem as long as the aged utilization is no greater than 50%.

To show the second part, consider the problem having a single task τ_1 where $C_1 = A_1/2$. It is always the case $L_i \ge C_i$ for any possible task set and scheduling algorithm, and the only feasible schedule for this problem dispatches the task with $T_1 = C_1$, which has 100% traditional utilization and 50% aged utilization. §

These results show how a scheduling algorithm for the traditional periodic task problem where deadlines equal periods (called implicit deadlines) can be used to solve the age scheduling problem. We will call this age scheduling by reduction to some traditional algorithm, e.g. age scheduling by reduction to rate monotonic scheduling, age scheduling by reduction to earliest deadline first scheduling.

The preceding results emphasize that the traditional and aged utilizations are different metrics that can both be computed for an age scheduling problem. The traditional utilization will in general vary depending on the scheduling algorithm, while the aged utilization is strictly a function of the problem specification. We now show that the aged utilization metric is always less than 100% even for the best possible schedule.

Defn: A *perfect* age schedule is one in which $L_i = C_i$ for all tasks.

A perfect schedule provides the smallest latencies that could

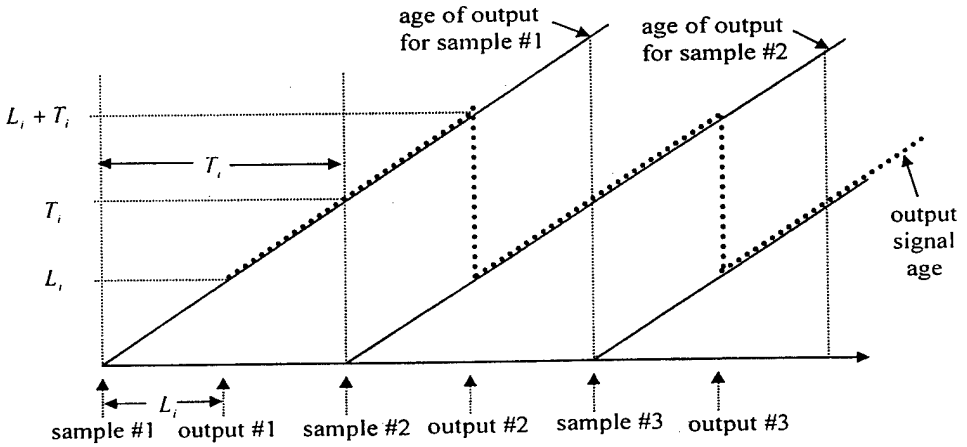

Fig. 3. Age of an output as a function of time.

possibly exist. Figure 4 shows an example of a perfect age schedule, one achieving a traditional utilization of 1 for a problem having an aged utilization of $2/3$. (This figure also illustrates a nice visualization of the age, which is the interval of time between a dispatch of a task and the completion of the following dispatch.) A perfect age schedule has the following properties (which we state without proof).

- A perfect age schedule is preemption-free and jitter-free.
- A perfect age schedule may have non-zero phasing or offset between different tasks (i.e. offsets are decided by the scheduling algorithm).
- A perfect age schedule achieves the least possible ratio U/U_A of traditional to aged utilization among all possible schedules for a given problem.

We now bound the value of U_A relative to U for perfect schedules, which bounds the aged utilization that could be achieved by any possible age scheduling algorithm. This will provide something analogous to the impossible-to-exceed limit on traditional utilization, $U \le 1$ for all possible feasible schedules. We first state two simple lemmas, then an approximate bounding theorem.

Lemma 3: $\frac{C}{T} = \left(1 + \frac{C}{T}\right)\left(\frac{C}{T+C}\right)$

Proof: Divide both sides by $\left(1 + \frac{C}{T}\right)$, you end up with $\left(\frac{C}{T+C}\right)$ on both sides. §

Lemma 4: If $\sum_{i=1..n} x_i = U$, then $\sum_{i=1..n} x_i^2$ is minimal when $x = U/n$, and the minimal value is U^2/n.

Proof: The author has it on good authority that this is obvious to anyone familiar with Lagrange multipliers[2]. §

Theorem 3: For a perfect age schedule of n tasks,

$$U - \frac{U^2}{n} < U_A < U - \frac{U_A^2}{n}.$$

Proof: The proof is by algebraic manipulation of the formula for U/U_A for a perfect schedule,

$$\frac{U}{U_A} = \frac{\sum_i C_i/T_i}{\sum_i C_i/T_i + C_i}$$

Using the lemma 3 above, this can be rewritten as

$$\frac{U}{U_A} = \frac{\sum_i C_i/T_i + C_i + \sum_i \left(C_i/T_i\right)\left(C_i/T_i + C_i\right)}{\sum_i C_i/T_i + C_i}$$

With a few more simple manipulations, this can be written in the form

$$U_A = U - \sum \left(C_i/T_i\right)\left(C_i/T_i + C_i\right)$$

We now resort to an inexact approximation to bound the true result, using the fact that $C/T > C/(T+C)$,

$$U - \sum \left(C_i/T_i\right)^2 < U_A < U - \sum \left(C_i/(T_i + C_i)\right)^2$$

Using lemma 4 above, we can rewrite this as

$$U - U^2/n < U_A < U - U_A^2/n$$

§

Applying the quadratic formula to the right-hand side allows us to plot bounds on the maximum possible U_A for any given U. Figure 5 shows this approximation for U=1, which defines a region in which the maximum possible U_A (the U_A of a perfect schedule for a problem having the most benign possible set of compute times) lies. Note that the most benign possible set of compute times are those for which all tasks have identical task utilization (i.e. a perfectly balanced

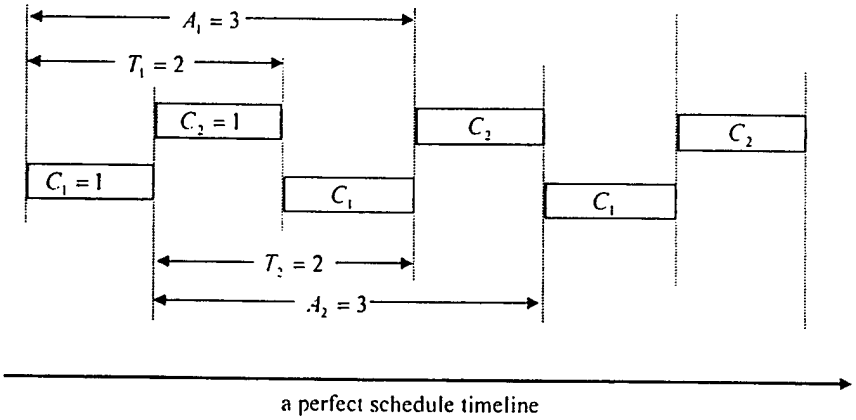

Fig 4 Example of a perfect age schedule.

workload).

Theorem 4: There exist feasibly schedulable problems for which no feasible perfect schedule exists (i.e. preemption and/or jitter may be necessary to feasibly schedule some workloads).

Proof: The system of two tasks τ_1, τ_2 with A_1=5, C_1=2, A_2=10, C_2=2 has no feasible non-preemptive age schedule. A feasible schedule must fit two executions of τ_1 into an interval of length 5 (the age is the interval between a dispatch of a task and the subsequent completion of the next dispatch), so any remaining idle intervals can have length at most 1. A preemptive fixed priority schedule, where τ_1 has priority over τ_2, T_1=3 and T_2=6, and the dispatches are offset by 2, is feasible and has U=1 and U_A=3/5=60%. §

What this theorem says is that the previous aged utilization ratio is still overly optimistic for determining an absolute not-to-exceed aged utilization bound for the general age scheduling problem. It is a bound that can be achieved by an algorithm only for particular age scheduling problems, those for which a feasible perfect schedule exists.

VI. Multi-processor age scheduling

The distributed age scheduling algorithms we consider take as input sets of flows, flow age constraints, and task compute times; and outputs an age constraint for every task along every flow. We formulate this as a constraint satisfaction problem in the following way.

We showed earlier that each flow age is the sum of the task ages along that flow. We also showed that each processor has an associated scheduling algorithm and aged utilization bound that we can use to assess schedulability. We thus have a set of constraints of the form

$$\text{for each flow } \phi,\ \sum_{I \in \Psi_\phi} A_I \le A_\phi$$

$$\text{for each processor } \rho,\ \sum_{I \in \Psi_\rho} \frac{C_I}{A_I} \le U^*_\rho$$

The distributed age scheduling problem is to find a set of task ages A_I that satisfy (at least) these constraints. In a later section we discuss a sample problem solved using an existing solver.

We now turn to the question of whether there is any inherent inefficiency in using a distributed asynchronous architecture versus a centralized synchronous one. Specifically, we evaluate a choice between n slower distributed processors versus a single central processor that is n times faster. Which choice can handle the larger workload while still meeting deadlines? For example, a flow through three perfectly scheduled asynchronous tasks, each having a compute time of 1 and a period of 2, has a worst-case age of 9 and loads the system to ½ capacity (assuming a classical breakdown utilization of 1). But a single processor that is 3 times faster can achieve the same worst-case age with a period of 8 and loads the processor to only 1/8 capacity. To try and shed insight on the general question, we now present a relationship between the distributed system utilization and the fast uni-processor utilization.

For a given flow ϕ through a set Ψ_ϕ of distinct tasks on distinct processors, consider the equation

$$\sum_{I \in \Psi_\phi} \frac{C_I}{A_I} = E \frac{\sum_{I \in \Psi_\phi} C_I}{\sum_{I \in \Psi_\phi} A_I}$$

The left-hand side is the sum of the utilizations on all the processors, which is the total utilization U_S of the distributed system (which may be greater than 1). The right-hand side can be recognized as the total work performed divided by the end-to-end flow age. This is the aged utilization for an idealized uni-processor that, to an external observer, does the same work with the same age. The factor E is the relative efficiency of the distributed to the uni-processor system. We

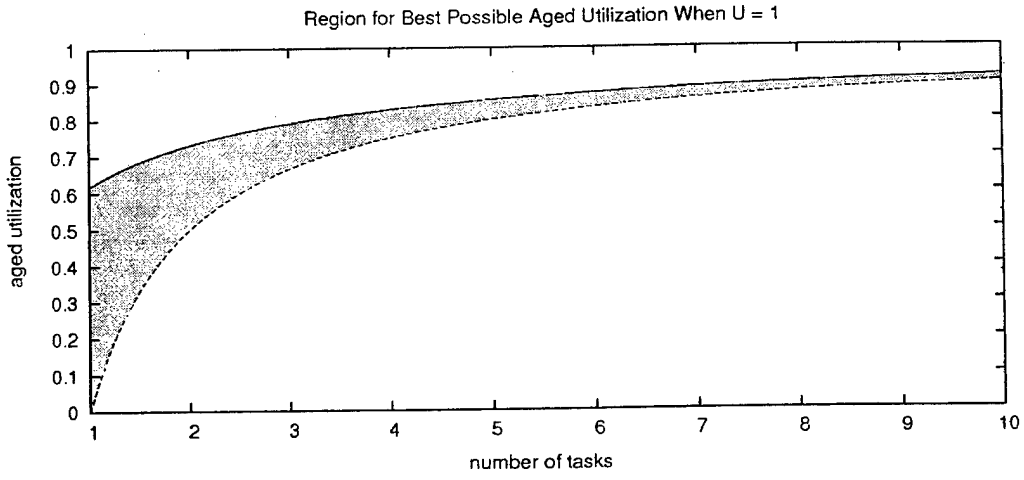

Fig. 5. Bounding region for best possible aged utilization when traditional utilization equals one..

now derive bounds on the ratio of these two values (bounds on the value of E) when the distributed system is scheduled as efficiently as possible (in the sense of minimum possible system utilization U_S).

Lemma 5: For a single flow ϕ the distributed system utilization

$$U_S = \sum_{i \in \Psi_\phi} \frac{C_i}{A_i}$$

is minimized when

$$\frac{C_j}{A_j^2} = \frac{C_k}{A_k^2}$$

for all j, k.

Proof: The lemma says that the sum of the individual processor utilizations is minimized when the derivatives of the individual processor utilization curves are equal. Figure 5, which shows the aged utilization curve as a function of the age for an individual processor, may be helpful.

Suppose the lemma were not true, and the sum was minimal when the derivatives with respect to the ages

$$\frac{\partial C_j / A_j}{\partial A_j} > \frac{\partial C_k / A_k}{\partial A_k}$$

were greater on processor j and smaller on processor i. Then a slight decrease in A_j and an equally slight increase in A_k would preserve the same end-to-end age while slightly reducing the sum of the utilizations, a contradiction. §

Lemma 6: For any list of n non-negative values $x_1..x_n$,

$$1 \le \frac{\left(\sum_i x_i\right)^2}{\sum_i (x_i)^2} \le n$$

(the ratio of the square of the sums to the sum of the squares never exceeds the number of values). The maximum value is achieved when $x_i = x_j$ (all the x values are identical). The minimum value is achieved when all but one x_i asymptotically approach zero.

Proof: We can use lemma 4 to show this, letting $\sum_{i=1..n} x_i = U$ and $\sum_{i=1..n} x_i^2 = U^2/n$. The ratio is maximized when the denominator is minimized. Lemma 4 and a little algebraic manipulation show the maximum ratio is n, and that this is achieved when $x_i = x_j$. For the minimum ratio, we note that a list of n numbers can always become dominated by a single value, where all but one of the values (say, x_k) asymptotically approach 0. In this case the ratio approaches $x_k^2 / x_k^2 = 1$. §

Theorem 5: For a single flow ϕ, if for all j and k

$$\frac{C_j}{A_j^2} = \frac{C_k}{A_k^2}$$

then

$$\frac{\sum_{i \in \Psi_\phi} C_i}{\sum_{i \in \Psi_\phi} A_i} \le \sum_{i \in \Psi_\phi} \frac{C_i}{A_i} \le |\Psi_\phi| \frac{\sum_{i \in \Psi_\phi} C_i}{\sum_{i \in \Psi_\phi} A_i}$$

Proof: It is fairly easy to see that

$$\sum_{i \in \Psi_\phi} \frac{C_i}{A_i} = \left(\frac{\sum_{i \in \Psi_\phi} A_i}{\sum_{i \in \Psi_\phi} C_i} \sum_{i \in \Psi_\phi} \frac{C_i}{A_i} \right) \frac{\sum_{i \in \Psi_\phi} C_i}{\sum_{i \in \Psi_\phi} A_i}$$

meaning the efficiency factor is

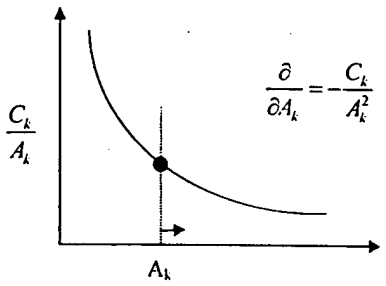

Fig. 5. Aged utilization as a function of age.

$$E = \left(\frac{\sum_{i \in \Psi_\phi} A_i}{\sum_{i \in \Psi_\phi} C_i} \sum_{i \in \Psi_\phi} \frac{C_i}{A_i} \right)$$

We show by algebraic manipulation that $E \le n$. We rewrite the right-hand sum using $\prod_i A_i$ as the common denominator,

$$\frac{\sum_i A_i}{\sum_i C_i} \left(\frac{C_1 \prod_{k \ne 1} A_k + C_2 \prod_{k \ne 2} A_k + \cdots}{\prod_i A_i} \right)$$

which, using slightly different notation can be written

$$\frac{\sum_i A_i \sum_i C_i \prod_{k \ne i} A_k}{\sum_i C_i \prod_i A_i}$$

We combine the sums in the numerator and separate out one age from each term in the denominator, we can rewrite this as

$$\frac{\left(\sum_l C_l \prod_{k\neq l} A_k\right)\left(\sum_l A_l\right)}{\sum_l \left(C_l A_l \prod_{k\neq l} A_k\right)}$$

We now make a substitution easily derived from the premise of the theorem, letting $C_l = (C_1/A_1^2)(A_l^2)$ in $\sum_l C_l \prod_{k\neq l} A_k$.

This leads to the following reductions

$$\begin{aligned}
&\sum_l C_l \prod_{k\neq l} A_k \\
&= \sum_l (C_1/A_1^2) A_l^2 \prod_{k\neq l} A_k \\
&= (C_1/A_1^2) \sum_l A_l^2 \prod_{k\neq l} A_k \\
&= (C_1/A_1^2) \sum_l A_l \prod_{k} A_k \\
&= (C_1/A_1^2) \prod_k A_k \sum_l A_l
\end{aligned}$$

If we apply this substitution to the previous equation, we get

$$\frac{\left(\sum_l C_l \prod_{k\neq l} A_k\right)\left(\sum_l A_l\right)}{\sum_l \left(C_l A_l \prod_{k\neq l} A_k\right)} = \frac{\left((C_1/A_1^2)\prod_k A_k\right)\left(\sum_l A_l\right)\left(\sum_l A_l\right)}{\left((C_1/A_1^2)\prod_k A_k\right)\left(\sum_l A_l A_l\right)}$$

We recognize this as the ratio of the square of the sum to the sum of the squares, which by lemma 5 is never less than 1 or greater than $|\Psi_\phi|$ (the length of flow ϕ, the number of distinct processors that host the flow). §

As noted in lemma 5, the minimum ratio above is achieved when all but a single A_l asymptotically approach zero. This occurs when the corresponding C_l values are asymptotically small. This is the case where the distributed flow asymptotically approaches a single processor workload anyway.

The maximum ratio is achieved when all the A_l values are equal, which occurs when the corresponding C_l values are equal, and in which case the utilizations on all the distributed processors are equal. With equal utilizations, and a single resource age scheduling algorithm having an aged breakdown utilization of A_A^*, the distributed system is feasible providing

$$\frac{1}{n}\sum_l \frac{C_l}{A_l} \le U_A^*$$

We assume the virtual centralized processor is n times faster than any single distributed processor, so that the centralized compute times take the form C_l/n. We assume an equally efficient single resource age scheduling algorithm is used. Then the virtual centralized processor is feasible providing

$$\frac{1}{n}\frac{\sum_l C_l}{\sum_l A_l} \le U_A^*$$

Theorem 5 tells us that the left-hand-side of the first inequality above can be as much as *n* times greater than the left-hand-side of the second inequality. In some sense, then, centralized synchronous scheduling can be as much as *n* times more efficient than asynchronous distributed scheduling for a single flow. The greatest discrepancy in efficiency occurs when the flow computation is evenly distributed across the distributed asynchronous system.

These observations are based on an analysis of a single flow. However, we note theorem 5 is independent of the utilizations, it is true of a flow that loads each processor very lightly as well as a flow that loads each processor to the breakdown utilization. This implies the distributed and centralized processor utilizations for multiple flows would sum independently, which suggests this result holds for multiple flows as well.

VII. AN EXAMPLE

An important practical question is whether a usefully large problem can be tractably solved using available methods for finding solutions to the system of non-linear constraints shown at the beginning of section V. To assess this, we used data derived from a planned avionics system to produce and solve such a system of constraints.

The core of the hardware architecture consisted of two computing clusters. Each cluster contained a high-speed time division multiple access bus, 4 central processors, and 2 external bus interface modules. Each external bus interface module connected to 4 low-speed time division multiple access busses, each of which connected to between 2 and 4 I/O processors. Each I/O processor connected to a large number of external devices (sensors and actuators).

The software architecture consisted of 40 dual-redundant software applications, each containing between 2 and 4 periodic tasks. There was also a software task to manage each external device, hosted on the connected I/O processor. The software architecture consisted of a total of 1472 periodic tasks and 2644 flows between external devices and application tasks.

Before generating the model, we merged all tasks that were bound to the same I/O processor and executed at the same rate into a single task. We merged (multiplexed) every set of flows that had common source and destination processors, destination tasks with identical periods, and identical routing through the hardware resources. This reduced the totals to 210 tasks and 610 flows.

In formulating the system of non-linear constraints that describe feasible system schedules, we must select a set of flows whose end-to-end age is to be constrained. In practice,

this will be derived by the developers based on individual application timing requirements. The only values available to us, however, were the sampling rates of the application tasks hosted on the central processors We used three times this value as our maximum allowed end-to-end age between external device and application function.

Each bus was treated as a hardware processor, and external bus interface modules were ignored. Thus, a flow consisted of a periodic task on an I/O processor, a periodic task on a low-speed bus, a periodic task on a high-speed bus, and a periodic task on a central processor, with an end-to-end age constraint equal to three times the specified sampling rate of the task on the central processor.

The resulting multi-processor age scheduling problem had 1425 variables and 1872 constraints. We assumed a maximum (breakdown) aged utilization of 80% for processors and 50% for busses. We automatically generated an AMPL model from a specification of this architecture written in (a preliminary version of) the SAE standard Avionics Architecture Description Language (AADL). AMPL expects a goal function to be optimized, we minimized the sum of the maximum processor aged utilization and maximum bus aged utilization.

This model was solved in about 45 seconds using CONOPT. The final solution had maximum processor aged utilizations of about 63% and maximum bus aged utilizations of about 43%.

VIII. Summary and Future Work

Age, as we have defined it here, has intuitive appeal as a metric for specifying end-to-end timing constraints in distributed asynchronous systems. However, we need to determine if this metric has a useful meaning for control or signal processing engineers. The approach to sampled data systems that is presented in introductory texts is to assume an algorithm is specified as a set of difference equations that are evaluated periodically and instantaneously. Under these assumptions, a transfer function for the sampled data subsystem can be derived, Shannon's sampling theorem can be used to relate the sampling rate to the frequency spectrum of the signal, etc. Additional work is needed to see how the concept of age (as defined here) manifests itself within the signal processing and control engineering domains, e.g. can a transfer function be derived when the end-to-end controller timing behavior is specified as a worst-case age (as defined here)?

The largest and smallest ages can vary significantly, which is to say an end-to-end flow may experience significant end-to-end jitter. Moreover, this may be fairly unstable in the sense the actual end-to-end latency may change a lot over a very short interval of time. This aspect of end-to-end timing behavior is worth further investigation.

We carried out some preliminary investigations into relationships between age scheduling and traditional real-time periodic scheduling and analysis. A similar investigation could be undertaken with respect to real-time switched network scheduling and analysis.

Improved uni-processor age scheduling algorithms certainly remain to be discovered. Each such algorithm should have an associated aged utilization bound that can be used during the distributed scheduling phase. The more precise the aged utilization bound, the more efficient the distributed scheduling solution will be. Such bounds may be made more precise by taking into account the compute times for the tasks hosted on a processor.

A near-term pragmatic problem is further work on age scheduling by reduction to legacy scheduling algorithms. For example, quantization of task periods needs further investigation. Methods to obtain estimated aged utilization bounds that are sufficiently precise to enable reasonably efficient distributed scheduling are needed. Note that these do not necessarily need to be analytic bounds, since the individual processor analytic methods can be used to verify the final solution to a high level of assurance.

Acknowledgments

The author would like to thank Larry Stickler, Denis Foo Kune and Ted Bonk for helping create the AADL architecture specification; and Nitin Lamba for his help with AMPL and CONOPT.

References

[1] Jean-Yves Le Boudec and Patrick Thiran. *The Network Calculus*. http://ica1www.epfl.ch/PS_files/NetCal.htm

[2] Daniel Johnson and Mike Elgersma. personal communication. Honeywell Labs, 2003.

[3] Jun Sun and Jane Liu, "Synchronization Protocols in Distributed Real-Time Systems," Proceedings of the 16th International Conference on Distributed Computer Systems, 1996.

[4] K. Tendell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," Microprocessors and Microprogramming 40, 1994.