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.

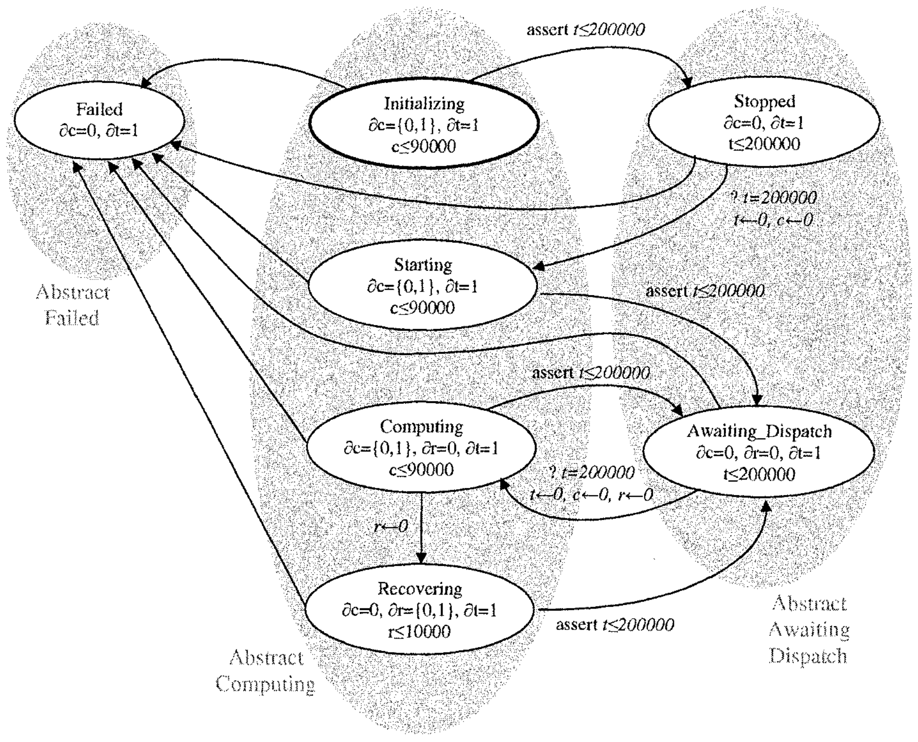

Figure 1. Concrete Hybrid Automata Model T for a MetaH Periodic Task

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.

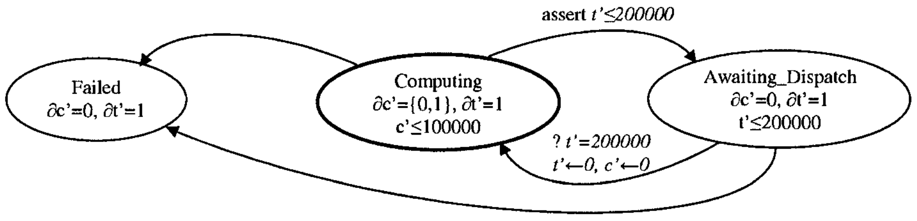

Figure 2. Abstract Hybrid Automata Model T' for a MetaH Periodic Task

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 location 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 $\delta r = 1/2$

d $\delta c = 0$ 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 Ta-

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

ble 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.

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

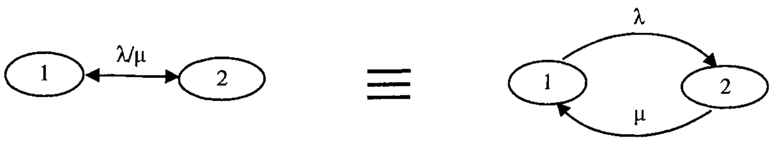

Figure 3. Failure/Repair Transition Notation and SRC Model

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.

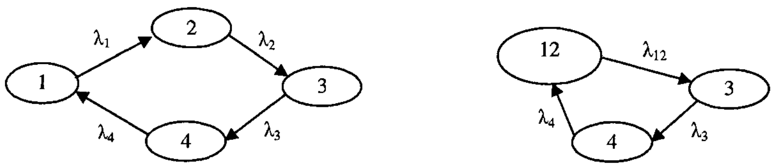

Figure 4. Markov Cycle Models (Right abstracts Left)

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

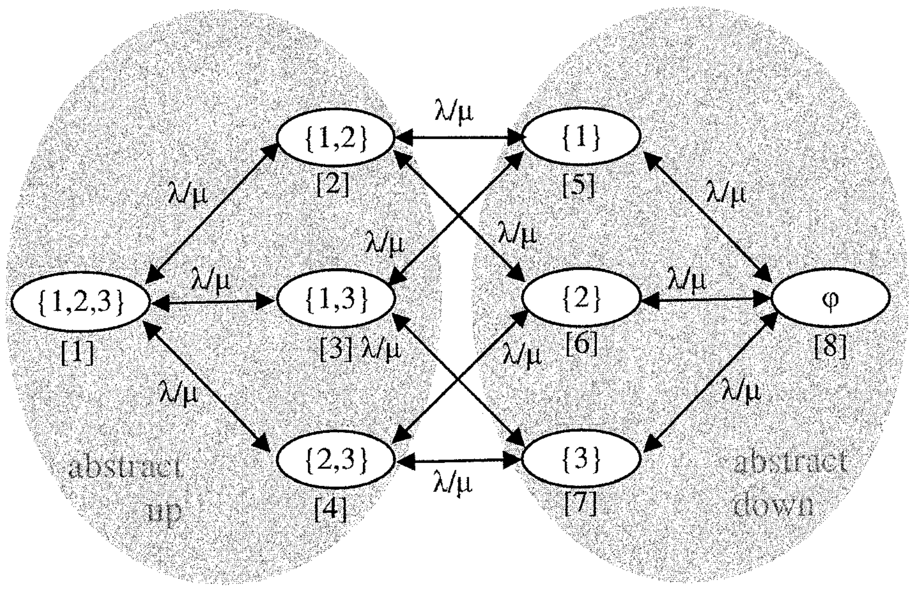

Figure 5. Markov TMR Model

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.

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

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

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

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

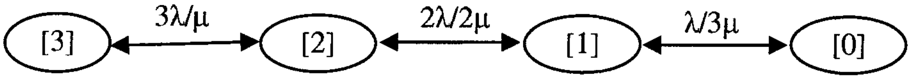

Figure 6. Number of Operational Components TMR Abstraction

(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.

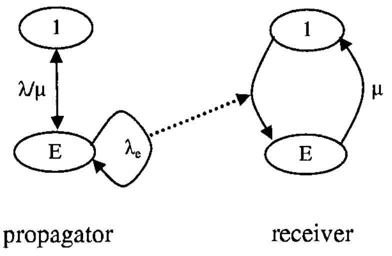

Figure 7. Error Propagation Between Markov Models

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.