In Proc. of the 11th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC 2008), 2008.

Compositional Feasibility Analysis of Conditional Real-Time Task Models *

Madhukar Anand, Arvind Easwaran, Sebastian Fischmeister and Insup Lee
Department of Computer and Information Science
University of Pennsylvania
{anandm,arvinde,sf,lee}@seas.upenn.edu

Abstract

Conditional real-time task models, which are generalizations of periodic, sporadic, and multi-frame tasks, represent real world applications more accurately. These models can be classified based on a tradeoff in two dimensions – expressivity and hardness of schedulability analysis. In this work, we introduce a class of conditional task models and derive efficient schedulability analysis techniques for them. These models are more expressive than existing models for which efficient analysis techniques are known. In this work, we also lay the groundwork for schedulability analysis of hierarchical scheduling frameworks with conditional task models. We propose techniques that abstract timing requirements of conditional task models, and support compositional analysis using these abstractions.

1 Introduction

Embedded real-time processes are typically implemented as some event-driven code embedded within an infinite loop. In many applications, the action to be taken upon the occurrence of external events depends on factors such as the current state of the system, values of external variables, etc. These systems can be modeled using well known task frameworks such as periodic/sporadic tasks, task graphs, and timed automata. Although schedulability is a well studied problem for periodic/sporadic tasks, they lack the expressivity of task graphs and timed automata. On the other hand, schedulability analysis for general task graphs and timed automata is hard.

A subclass of task graphs, called recurring branching tasks [3], are more expressive than periodic/sporadic tasks and their demand computation for schedulability analysis can be done efficiently. In this work, we introduce new models that are strictly more expressive than the recurring task model used by Baruah [3], but at the same time retain their efficiency of demand computation. Specifically, we support analysis for models whose period of recurrence for different branches is not identical (anisochronous), and for models with explicit control variables, transition guards, and assignments. We call these models Recurring Branching Task Model (RBT) and Recurring Task Model with Branching and Control variables (RTC) respectively.

The increasing complexity of real-time embedded systems requires advanced design and analysis methods for the assurance of timing requirements. Component-based techniques have been widely used to design such systems. They often involve hierarchical scheduling frameworks with conditional task models for supporting hierarchical resource sharing under different policies. To take advantage of component-based design, schedulability analysis of hierarchical frameworks must then be addressed. Furthermore, to be faithful to the paradigm of component-based engineering, it is desirable to do this analysis *compositionally*, i.e., system-level schedulability analysis should be done by combining component interfaces that abstract component-level timing requirements. In this work, we develop techniques to abstract the resource demand of RBT/RTC models into interfaces, and thus address compositional analysis for these models.

For real-time systems, there has been a growing attention to compositional analysis of hierarchical scheduling frameworks [6, 10, 11, 8, 13, 9, 14]. Traditionally, this analysis has been achieved using resource model based interfaces that characterize the resource supply necessary to schedule components. Mok and Feng proposed the bounded-delay resource partition model for a hierarchical scheduling framework [6], and Shin and Lee [13] addressed the interface generation problem for these models. Similarly, there are studies [11, 8, 12] on the component abstraction problem using periodic resource models. When these techniques are used for hierarchical frameworks with conditional task models, complexity of interface generation depends on the utilization of task set (i.e., schedulability checking of conditional task models depends on utilization [3]). Specifically, if the utilization is high, then these procedures are inefficient. On the other hand, in this paper, we directly synthesize component interfaces from the resource demand of conditional task models in those components. Our tech-

*This research was supported in part by FA9550-07-1-0216 NSF CNS-0509143, NSF CNS-0721541, NSF CNS-0720703, and OEAW APART-11059

nique is independent of task set utilization, and therefore does not suffer from the same drawback as resource model based techniques.

In other related work, Thiele *et al.* [14], and Matic and Henzinger [9], introduced real-time interfaces for the compositional analysis problem. However, these approaches cannot be applied to hierarchical frameworks with conditional task models. Recurring branching task models and their schedulability analysis was introduced by Baruah [4, 3]. Bordoloi and Chakraborty [5] extended this analysis for models with dynamic task parameters. However, both these approaches consider restricted models and also do not address the compositional analysis problem.

1.1 Motivational Example

Consider the example of the Three Tanks System (3TS) [7] shown in Figure 1. The plant consists of interconnected water tanks, where each tank has evacuation taps for simulating perturbation. Two of the tanks (T1 and T2) can pump water into other tanks via the pumps P1 and P2. The plant is nonlinear and hence it uses three different controllers for each pump. (1) A controller P (proportional) is used for the case when there is no perturbation (no water leaves the tank). (2) A controller PI (proportional integrator) is used when there is some perturbation (water drains out of the tank). When the control error is large, a controller with fast integration speed is used and otherwise, a controller with slow integration speed is used.

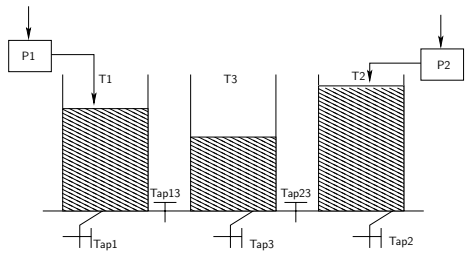


Figure 1. Overview of 3TS

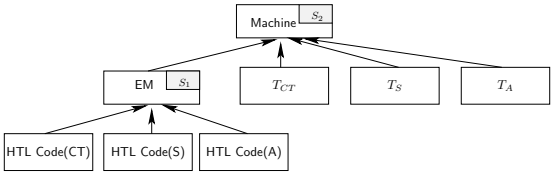


Figure 2. System hierarchy

Figure 1.1 shows the scheduling hierarchy of the 3TS system. At the first level, we have the controller (CT), sensor (S) and actuator (A) switching logic captured as HTL code and scheduled by the virtual machine (EM). Each of these modules release tasks T_{CT}, T_S and T_A, respectively, which actually perform the control, sensing and actuation. These tasks are written in a high level language like C and are scheduled by the operating system. In our example, we assume that EM uses EDF (S_1=EDF) for scheduling the HTL tasks, and the operating system uses RM (S_2=RM) to schedule tasks at the second level. Figure 1.1 then shows the conditional models for modules in 3TS (c.f.,HTL code [1]).

As can be seen from this example, in practice, embedded real-time systems comprise of multiple components, each modeled using conditional tasks. Therefore, as discussed before, there is a need for compositional analysis techniques for such complex systems. The abstraction techniques we present in this paper, aim to fill this gap.

2 Task Model and Definitions

In this section, we define our recurring task models and their execution semantics. Our system consists of multiple real-time components sharing a global resource (e.g., CPU, shared network, etc.) under a hierarchical scheduling policy. The shared resource demand of each component can be represented by a set of RBT models, each comprising of multiple simple tasks as the basis for demand. A *simple task* $T = (e,d)$ requires e time units of the resource within d time units of its release.

Informally, a RBT model is a structure consisting of nodes and transitions between these nodes, where each node defines a release of a simple task and each transition identifies the minimum jitter between successive task releases.

Definition 1 (RBT Model) *A RBT model* Ω *is a tuple* $\langle V, v^0, V^F, E, \tau, \rho \rangle$ *where,*

- V *is a set of nodes,*
- $v^0 \in V$ *is the start node,*
- $V^F \subseteq V$ *is a set of final nodes called leaves,*
- $E \subseteq V \times V = E_T \cup E_R$ *is a set of transitions where* E_R *is a set of resets,*
- $\tau : V \rightarrow \mathcal{T}$ *is a function from nodes to simple tasks,*
- $\rho : E \rightarrow \mathbb{N}$ *is a function from a transition to minimum jitter.*

$E_R = \{\langle v, v^0 \rangle | v \in V^F\}$ *and* $E_T = E \setminus E_R$ *such that the underlying graph* (V, E_T) *is a directed tree.*

For this model, we assume that any node releases one simple task. Multiple task releases can be handled by transitions with zero jitter. The execution semantics of a RBT model may be described as follows. The execution starts at node v^0 where the task $\tau(v^0)$ is released. After the release, a transition from v^0 is chosen non-deterministically to one of the descendant nodes of v^0 (say, v). This transition is taken after a minimum delay of $\rho(\langle v^0, v \rangle)$, and this process of task release continues from node v. We now introduce some definitions related to the RBT model.

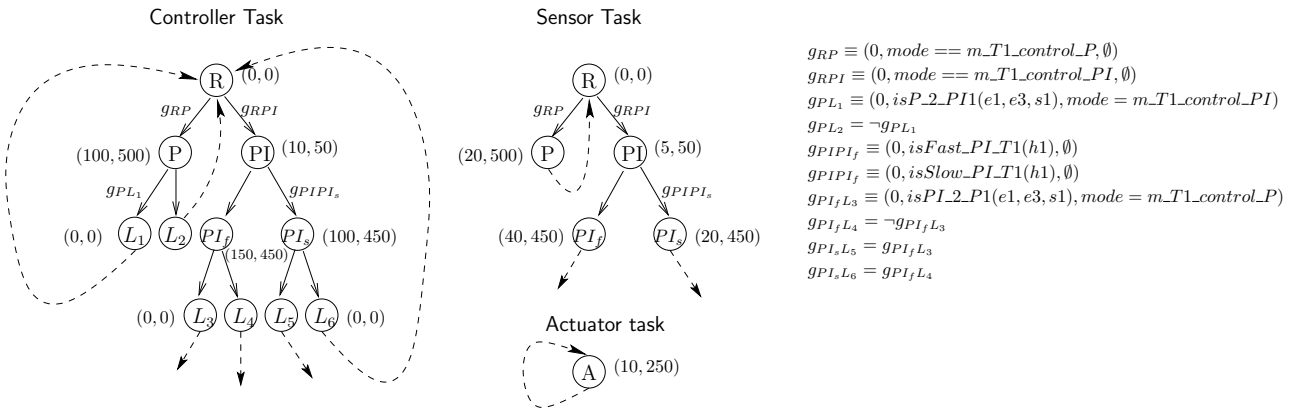


Figure 3. Conditional models for EM in the 3TS system

Definition 2 (Run) *A run $r \equiv run(v^i, v^{i+j}, t)$ is a sequence of progression of nodes from v^i to v^{i+j} of a RBT model $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$: $v^i \xrightarrow{e^{i+1}} v^{i+1} \xrightarrow{e^{i+2}} \ldots \xrightarrow{e^{i+j}} v^{i+j}$ where $\forall l \in [1,j]$, $e^{i+l} = \langle v^{i+l-1}, v^{i+l} \rangle \in E$ and $t = \sum_{k=1}^{l} \rho(e^{i+k})$. We denote by $\Gamma(r)$ the duration t of run r. Also, the* resource demand *of the run r is defined as $\Delta(r) = \sum_{l=0}^{j} \tau(v^{i+l})$.e.*

In this definition $\tau(v^{i+l}).e$ represents the execution requirement of task $\tau(v^{i+l})$. Now we can formally define the term isochronous and anisochronous RBT models.

Definition 3 (Isochronicity) *A RBT model Ω is* isochronous *if $\forall v^i, v^j \in V^F, \Gamma(run(v^0, v^i, t)) + \rho(v^i, v^0) = \Gamma(run(v^0, v^j, t)) + \rho(v^j, v^0)$. In this case, the smallest t for which this condition is true, is called the period of Ω. In all other cases, Ω is* anisochronous.

For an isochronous RBT model Ω with period P, we define the worst case loop, $wcl(\Omega) = \arg\max_r \Delta(r)$ where $r \equiv run(v^0, v^0, P)$. Intuitively, wcl_Ω is the loop with largest demand starting from v^0, ending at v^0 and passing through exactly one leaf.

Definition 4 (RTC Model) *The RTC model Ψ is similar to the RBT model Ω (Definition 1), except that ρ is now defined as a function from a transition to minimum jitter, an enabling condition, and a variable assignment ($\rho : E \rightarrow \mathbb{N} \times G \times A$).*

$a \in A$ consists of assignment for variables in $\mathcal{V}$ and $g \in G$ is any decidable function over the variables $\mathcal{V}$. The definitions of run, Γ, Δ, isochronicity, and *wcl* for RTC models are similar to those for RBT models, except that the transitions in a valid run must be enabled. Hence a run is only defined under a fixed variable assignment. In the remainder of the paper, although we use the same notation to denote a run ($run(v^a, v^b, t)$), we assume, strictly for didactic purposes, that there is an implicit initial variable assignment that uniquely identifies this run. The execution semantics of the RTC model are similar to the RBT model. In addition, the enabling conditions/variable assignments on a transition from v_i are assumed to be evaluated/executed immediately after the release of task $\tau(v_i)$, which is instantaneous. The 3TS system models (Figure 3) is an example of RTC model.

We make the following assumptions for a RTC model (1) the set of enabling conditions $g_1, \ldots, g_m$ on transitions leaving a node must be exhaustive, i.e., $\bigvee_{j=1}^{m} g_j = \text{true}$ (progress). (2) the enabling conditions and assignments have no overhead in terms of space and time. This assumption simplifies presentation of the paper, and the overhead can be easily integrated into our analysis, and (3) the set of leaf nodes V^F is nonempty, and every other node has a run to one of the leaf nodes.

3 Demand for anisochronous RBT models

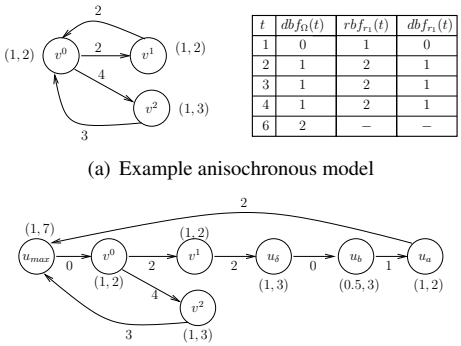
The resource demand bound function ($\text{dbf}_\Omega : \mathbb{R} \rightarrow \mathbb{R}$) of a RBT model Ω upper bounds the amount of computational resource required to meet the deadlines of all the released tasks. For a time interval length t, $\text{dbf}_\Omega(t)$ gives the largest resource demand of Ω in any time interval of length t. This computation is done over tasks that are both released and have their deadlines within the interval. Further, a run r of Ω such that $\Delta(r) = \text{dbf}_\Omega(t)$ is called a critical run for interval length t. Since we do not assume frame separation[1], the duration of this critical run need not be t. However, in the remainder of the paper, for clarity of presentation we will abuse notation in that we denote by t, the duration of a critical run for interval length t. The request bound function ($\text{rbf}_\Omega : \mathbb{R} \rightarrow \mathbb{R}$) of a RBT model Ω, upper bounds the amount of resource demand released in a time interval. The rbf computation takes into account the demand of all the tasks that are released in the interval, including those tasks whose deadlines are outside the interval.

[1] Under frame separation, the deadline of a task at any node is at most the minimum jitter over all outgoing transitions from that node. c.f., [3]

For isochronous RBT models, Baruah [3] has presented an efficient algorithm to compute the dbf. We summarize this technique below. Consider model $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$, and a time interval of length $t < 2P$ where P is the period of Ω. In any run of Ω with duration t, the start node v^0 occurs at most once. These runs can therefore be enumerated to compute dbf_Ω for all $t < 2P$. For $t \geq 2P$, the run with largest demand consists of three phases. (1) a $run(v^i, v^0, t_1)$ s.t. $t_1 < P$, (2) some $k \in \mathbb{N}$ multiples of $wcl(\Omega)$ of total duration t_2, and (3) a $run(v^0, v^j, t_3)$ s.t. $t_3 < P$ and $t_1 + t_2 + t_3 = t$. Since $run(v^i, v^0, t_1)$ ends in v^0 and $run(v^0, v^j, t_3)$ starts from v^0, we can concatenate them into a single run of duration $t_1 + t_3 (< 2P)$ for the purposes of dbf computation. Therefore, for all $t \geq 2P, \text{dbf}_\Omega(t) = \text{dbf}_\Omega(t_1 + t_3) + k\Delta(wcl(\Omega))$, where $\text{dbf}_\Omega(t_1 + t_3)$ is computed using the aforementioned dbf procedure for $t < 2P$.

The above procedure cannot be directly applied to anisochronous models. This is because the minimum duration between successive invocations of the start node in the anisochronous case, depends on the particular run. In fact, a reduction from the *integer knapsack* problem can be used to prove that dbf computation for this case is NP-hard. We now describe a procedure to transform anisochronous RBT models into isochronous models. This procedure ensures that the demand of the transformed model is at least as much as the demand of the anisochronous model. Consider an anisochronous RBT model $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$, such that $\langle p_1, \ldots, p_n \rangle$ represents the minimum durations of runs between successive invocations of the start node, through different leaf nodes. Furthermore, w.l.o.g we assume that $p_1 \leq p_2 \leq \ldots \leq p_n$. We denote by v^i and r_i, the leaf and run, respectively, that correspond to p_i. For example, consider the anisochronous RBT model shown in Figure 4(a). It consists of two different runs between successive invocations of the start node; one through leaf v^1 and another through v^2. In this case, the minimum durations of these runs are $\langle p_1 = 4, p_2 = 7 \rangle$.

We transform Ω to an isochronous model with period P for some $P \geq p_n$. For this transformation, we assume that for any $v \in V$, the deadline of $\tau(v)$ is at most the duration of the shortest run to v^0 from v (*reset frame separation*). This transformation procedure is presented in Algorithm 1. In the algorithm, the nodes labeled u, u_δ and u_{max} denote the inserted nodes. To make the presentation simple, in all the algorithms described in this paper, we assume that the creation of nodes and edges also appropriately update sets V, V^F, and E. In Lines 5-21 of the algorithm, we insert nodes between v^i and v^0 in Ω such that the duration of the inserted run is $P - p_i$. This inserted run mimics the transition jitters of the old run r_i (of duration $P - p_i$), but the demand of each inserted node is dbf_Ω for an interval length equal to the jitter on outgoing transition. This insertion ensures that any critical run in the old model also exists in the transformed model, and has at least the same demand. In the case that $p_i < P - p_i$, the introduced portion is longer than the old run, so the remainder demand for $P - 2p_i$ is inserted in Line 18.



t	$dbf_\Omega(t)$	$rbf_{r_1}(t)$	$dbf_{r_1}(t)$
1	0	1	0
2	1	2	1
3	1	2	1
4	1	2	1
6	2	–	–

(a) Example anisochronous model

(b) Transformed model

Figure 4. Model transformation

As we make the RBT model isochronous by introducing new nodes, it is possible that some of the older nodes get shifted out of a critical interval in the dbf computation. In Line 23 of the algorithm, we add a node that compensates for the demand of such displaced nodes. Finally, we introduce a node u_{max} at the beginning of the RBT which has a demand equal to the maximum demand over all inserted nodes (except u_δ and node inserted in Line 18). This is required to handle the case where the critical interval does not end on an inserted node. Whenever $P < 3p_1$, all values of dbf and rbf required by the algorithm can be computed in $O(|V|^3)$ time [3, 4], giving an overall running time of $O(|V|^3 + |V|P)$. In addition, we observe that (1) number of nodes inserted by the algorithm is $O(|V^F||V|)$ and (2) reset frame separation property is preserved. We illustrate the above algorithm using the anisochronous model shown in Figure 4(a) when $P = p_2 = 7$. The transformed isochronous model is shown in Figure 4(b).

Theorem 1 *Let $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$ and P be the input to Algorithm 1, and $\Omega' = \langle V', v^{0'}, V^{F'}, E', \tau', \rho' \rangle$ denote its output. Then, for all $t > 0, \text{dbf}_{\Omega'}(t) \geq \text{dbf}_\Omega(t)$.*

Due to lack of space, we provide proofs for all the theorems in our technical report [2]. We now present an upper bound on the demand overhead incurred in the conversion technique given by Algorithm 1. If the original anisochronous model is Ω and the corresponding isochronous model is Ω', the overhead is defined as follows. *Utilization overhead :*

$$UO_{(\Omega,\Omega')} = \max_{t>0} \frac{\text{dbf}_{\Omega'}(t)}{t} - \max_{t>0} \frac{\text{dbf}_\Omega(t)}{t}$$

This overhead reflects the increase in utilization that a resource supply would have to support due to the transformation. First, we make the following observation about dbf_Ω in any interval. This observation follows from reset frame

Algorithm 1 RTB-ISO-GEN(Ω, P)

Input: $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$, $p_1 \leq \ldots \leq p_n \leq P$
Output: Isochronous model Ω

1: Compute $\text{dbf}_\Omega(t), \forall t \leq \max_i\{P - p_i\}$. Let $d_{max} = 0$.
2: Compute $M = \max_{t<2p_n} \frac{\text{dbf}_\Omega(t)}{t}$.
3: **for** $i = 1$ to n **do**
4: Let $u_1 = v^i, e_o = \langle v^i, v^0 \rangle, u_2 = v^0, j = t = \rho(e_o)$
5: **while** $(t \leq P - p_i) \wedge (u_1 \neq \perp)$ **do**
6: Create node u s.t. $\tau(u) = (M \cdot \rho(e_o), t)$
7: $d_{max} = \max\{d_{max}, \tau(u).e\}$
8: Create transition $e = \langle u, u_2 \rangle$ s.t. $\rho(e) = \rho(e_o)$
// If $u_1 \neq v^0$, then PRED(u_1) returns predecessor of u_1, else it returns $\perp$.
9: $u_2 = u, e_o = \rho(\text{PRED}(u_1), u_1), u_1 = \text{PRED}(u_1)$
10: $t = t + \rho(e_o)$
11: **end while**
// The following condition checks for $p_i \geq P - p_i$
12: **if** $u_1 \neq v^0$ **then**
13: Create u, $\tau(u) = (M \cdot (P - p_i - t), P - p_i)$
14: $d_{max} = \max\{d_{max}, \tau(u).e\}$
15: Create $e = \langle u, u_2 \rangle$ s.t. $\rho(e) = P - p_i - t$
16: $j_\delta = 0$
17: **else**
18: Create u, $\tau(u) = (M \cdot (P - 2p_i), p_i)$
19: Create $e = \langle u, u_2 \rangle$ s.t. $\rho(e) = 0$.
20: $j_\delta = p - 2p_i$.
21: **end if**
22: Let $\delta = \max_{t<p_i}\{\text{rbf}_{r_i}(t) - \text{dbf}_{r_i}(t)\}$
23: Create u_δ, $\tau(u_\delta) = (\delta, P - p_i)$
24: Create $e_1 = \langle u_\delta, u \rangle$, $\rho(e_1) = j_\delta, e_2 = \langle v^i, u_\delta \rangle$, $\rho(e_2) = j$
25: **end for**
26: Create u_{max}, $\tau(u) = (d_{max}, P)$
27: **for** $v^i \in V^F$ **do**
28: Create $e = \langle v^i, u_{max} \rangle$, $\rho(e) = \rho(\langle v^i, v^0 \rangle)$
29: **end for**
30: Create $e = \langle u_{max}, v^0 \rangle$ with $\rho(e) = 0$, and let $v^0 = u_{max}$

separation property, and arguments similar to dbf computation for isochronous RBT models [3].

Proposition 1 *Given an anisochronous model Ω with reset frame separation property and a time interval t, we can partition t into sub-intervals $t_1, \ldots, t_m$ where $t_1, t_m \leq p_n$, and for $i = 2, \ldots, m-1$ $\exists j$, $1 \leq j \leq n$, with $t_i = p_j$, such that $\text{dbf}_\Omega(t) \geq \sum_{i=2}^{m-1} \max_{r_i} \Delta_\Omega(r_i) + \max_{r_1, r_m} \Delta(r_1 + r_m)$. Further, the run of length t_1 ends at v^0 and the run of length t_m begins at v^0.*

We now bound the time interval t up to which we need to check for computing $UO_{(\Omega,\Omega')}$.

Theorem 2 *Let $\Omega' = \text{RTB-ISO-GEN}(\Omega, P)$. Then,*

$$UO_{(\Omega,\Omega')} \leq \max_{t<2p_n} \frac{\text{dbf}_{\Omega'}(t)}{t} - b_\Omega - \max_{t<2p_n} \left\{ \frac{\max_{r \in \mathcal{R}} \Delta(r) - d_\Omega}{t} \right\}$$

where $\mathcal{R}$ is the set of all runs of Ω through v^0 and of duration less than $2p_n$ and $b_\Omega = \max_i \frac{\Delta(r_i)}{p_i}$ and $d_\Omega = 2p_n b_\Omega$.

We note that since Ω' is isochronous, computation of $\text{dbf}_{\Omega'}$ for values up to $2p_n$ is straightforward.

3.1 Demand for RTC models

Given an RTC model Ψ, we denote by dbf_Ψ its demand bound function, and by rbf_Ψ its request bound function. For a recurring real-time task model, Baruah [4] has given a technique for dbf computation which has exponential complexity, even when the model is isochronous. This complexity arises from the fact that between any pair of nodes there can be exponentially many runs that need to be considered. Note that a similar procedure, which takes into account enabling conditions on transitions, can be used for computing the dbf of RTC models. Without any restriction on the RTC model, this procedure also has exponential complexity.

To make dbf computation efficient, one way to restrict a RTC model is to ensure that any two nodes in the model have at most a constant number of *simple runs* between them. We denote this property as *constant simple runs* property. A simple run is a run that only uses transitions in E_T and no transition is repeated [2]. Under this restriction, a straightforward extension of the technique for RBTs can be used to compute the dbf for RTC models. It is worth noting that this restriction may not be necessary, and the dbf may be computable efficiently under other assumptions.

For anisochronous RTC models, a similar procedure as RTB-ISO-GEN (RTC-ISO-GEN) can be defined to transform them into isochronous RTC models. RTC-ISO-GEN is different from RTB-ISO-GEN in the following aspects. (1) PRED in Line 9 of RTB-ISO-GEN takes a run and a node and returns predecessor of the node in this run, (2) existing assignments and enabling conditions on transitions are preserved, and (3) the assignments and enabling conditions on new nodes are $\emptyset$. For correctness of this procedure, it is required that model Ψ satisfy reset frame separation property. Additionally, RTC-ISO-GEN only introduces $O|V|$ new nodes in the model as control variables can be used to merge common prefixes of final nodes (e.g., nodes introduced in Line 6 of RTB-ISO-GEN).

4 RBT Abstraction

In this section, we describe a technique to abstract a collection of RBT models into one RBT model. Specifically, given models $\Omega_1, \ldots, \Omega_m$, we develop a RBT abstraction (model) Ω such that $\forall t > 0, \text{dbf}_\Omega(t) \geq \sum_{i=1}^m \text{dbf}_{\Omega_i}(t)$. Informally, in Ω we introduce loops from the start node such that their demand satisfies the total dbf of models $\Omega_1, \ldots, \Omega_m$. The procedure for generating RBT abstraction is given in Algorithm 2. Firstly, we transform each model Ω_i to an isochronous model of period $P_m = P$ (Line 1). We add a

[2] Since a run in RTC model is defined under a fixed variable assignment, this is not overtly restrictive in that, it allows for multiple simple runs between a pair of nodes, each enabled by mutually exclusive constraints.

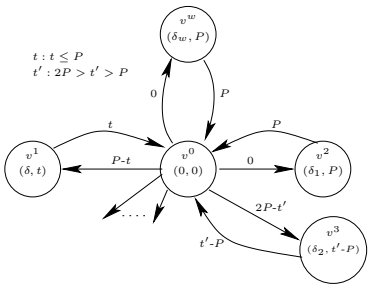


Figure 5. RBT abstraction

loop from the start node having demand equal to the concurrent execution of all $wcl(\Omega_i)$ (Line 2). This is shown in Figure 5 as the loop through node v^w. Now for each $t \leq P$, we add a node with demand $\delta = \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}(t)$ (Lines 4-8). This is shown in Figure 5 as the loop through v^1. Further, for each $P < t < 2P$, we add two nodes v^2 and v^3 that release tasks (δ_1, P) and $(\delta_2, t-P)$ as shown in Figure 5 (Lines 9-14), such that $\delta_1 + \delta_2 = \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}(t)$. We observe that the abstraction is isochronous, reset frame separated, and its size is $O(P \log P)$. The following theorem shows that Algorithm 2 generates a sound abstraction with respect to scheduling feasibility.

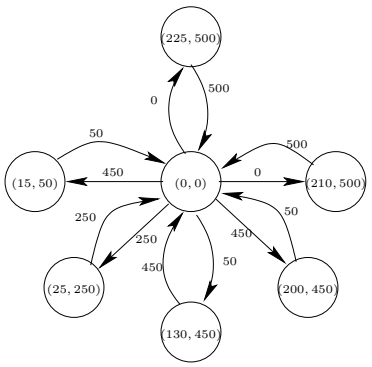
Algorithm 2 Algorithm for generating RBT abstraction

Input: $\Omega_1, \ldots, \Omega_m$,
where $\Omega_i = \langle V_i, v_i^0, V_i^F, E_i, \tau_i, \rho_i \rangle$.
Input: $P_1 \leq \ldots \leq P_m = P$, where P_i is period of Ω_i.
Input: $C = \{t < 2P | \forall \varepsilon > 0, \exists i, \mathrm{dbf}_{\Omega_i}(t) > \mathrm{dbf}_{\Omega_i}(t-\varepsilon)\}$
Output: $\Omega = \langle V, v^0, V^F, E, \tau, \rho \rangle$, s.t. $\mathrm{dbf}_{\Omega} \geq \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}$.

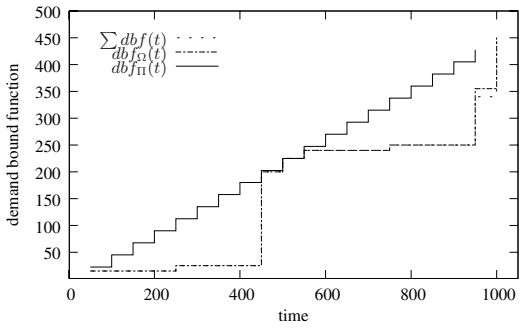
1: For each i, let $\Omega_i \leftarrow$ RTB-ISO-GEN(Ω_i, P)
2: Create v^0, $\tau(v^0) = (0,0)$; v^w, $\tau(v^w) = (\delta_w, P)$.
// $\delta_w = \sum_{i=1}^{m} \Delta(wcl(\Omega_i)) - \tau(v^0).e$.
3: Create $e_1 = \langle v^0, v^w \rangle$, $\rho(e_1) = 0$; $e_2 = \langle v^w, v^0 \rangle$, $\rho(e_2) = P$.
4: **for** $t \in C \wedge t \leq P$ **do**
5: Create v^1 s.t. $\tau(v^1) = (\delta, t)$ where $\delta = \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}(t)$
6: Create $e_1 = \langle v^0, v^1 \rangle$ s.t. $\rho(e_1) = P - t$.
7: Create $e_2 = \langle v^1, v^0 \rangle$ s.t. $\rho(e_2) = t$.
8: **end for**
9: **for** $t \in C \wedge (P < t < 2P)$ **do**
10: $\delta_1 = \sum_{i=1}^{m} \Delta(run(v_i^a, v_i^b, P))$
// **where** $run(v_i^a, v_i^b, P)$ **is a prefix of the critical run of** Ω_i **for interval length** t.
11: $\delta_2 = \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}(t) - \delta_1$
12: Create u_2, $\tau(v^2) = (\delta_1, P)$; v^3, $\tau(v^3) = (\delta_2, t-P)$.
13: Create $e_1 = \langle v^0, v^2 \rangle$, $\rho(e_1) = 0$; $e_2 = \langle v^2, v^0 \rangle$, $\rho(e_2) = P$; $e_3 = \langle v^0, v^3 \rangle$, $\rho(e_3) = 2P - t$; $e_4 = \langle v^3, v^0 \rangle$, $\rho(e_4) = t - P$.
14: **end for**

Lemma 1 *For all $t > 0$, $\mathrm{dbf}_{\Omega}(t) \geq \sum_{i=1}^{m} \mathrm{dbf}_{\Omega_i}(t)$.*

Theorem 3 *If RBT Ω can be feasibly scheduled on an uniprocessor platform, then RBT's $\Omega_1, \ldots, \Omega_m$ are also feasible to be scheduled.*



(a) RBT abstraction Ω



(b) dbf of the abstractions

Figure 6. Abstractions for 3TS

This result follows from Lemma 1.

Abstraction for 3TS. We illustrate this abstraction technique on 3TS introduced in Section 1.1. For the models shown in Figure 1.1, we ignore the guards and assignments on transitions. Then, these models are isochronous RBTs with period $P = 500$. The RBT abstraction Ω for EM (Figure 1.1) is given in Figure 6(a) and its dbf plotted in Figure 6(b). As can be seen, dbf_{Ω} closely follows the total dbf of underlying modules. For comparison, we show the abstraction of the 3TS modules using a periodic task with period 50.

5 RTC Abstraction

In this section, we develop a RTC abstraction for a set of RTC models. First, we describe a procedure to merge the critical runs from underlying models which is used in the abstraction procedure.

5.1 Merging runs of RTC models

Consider m runs $r_1, \ldots, r_m$ of RTC models, that are of the same duration t. Procedure 3 (MERGE) describes a technique to merge these runs into a single run r of duration t and demand $\Delta(r) = \sum_{i=1}^{m} \Delta(r_i)$, i.e., it generates a run whose

demand is equal to the demand of all runs r_i executing concurrently. In this procedure, we first insert all nodes in runs $r_1, \ldots, r_m$ into a min-priority queue ordered by the duration of partial runs leading to the node. Then, we extract each node v from this queue and insert it in a run r with an incident transition e from the current node v_c of r. e has the following properties: (1) the enabling condition on e checks if variable l_{v_c} is set to v, (2) the minimum jitter of e is such that the duration up to node v is preserved, i.e., if v belonged to a run r_i, then the duration of the partial run leading to v in r_i is preserved in r. Furthermore, the assignment on the transition incident on v_c is set to $l_{v_c} = v$. These assignments and enabling conditions on transitions will be used in the abstraction we generate, to prevent spurious runs. The merging procedure is demonstrated in Figure 7 where we only show the minimum jitter on transitions. In the figure, runs $r_1 = run(v^a, v^c, 9)$ and $r_2 = run(v^1, v^3, 9)$ are merged to give run $r = run(v^a, v^3, 9)$.

Procedure 3 MERGE$(r_1, \ldots, r_m)$

Input: Runs $r_1, \ldots, r_m$ s.t. $\forall i, j : \Gamma(r_i) = \Gamma(r_j)$.
Output: Run r s.t. $\Delta(r) = \sum_{i=1}^{m} \Delta(r_i)$ and $\Gamma(r) = \Gamma(r_i)$.

1: Create a *min-priority queue* $Q \leftarrow \emptyset$.
2: **for** $i = 1$ to m **do**
3: **for** each node v in r_i **do**
4: INSERT(Q, t_v) where $t_v = \Gamma(r_i^v)$, r_i^v being the partial run of r_i leading to node v.
 // We assume that node v **is inserted into** Q **as satellite data.**
 // We also assume that in case of conflict, nodes of r_i **have higher priority than nodes of** r_j **for all** $i < j$**.**
5: **end for**
6: **end for**
7: Let $v_c = \text{EXTRACT-MIN}(Q)$, $e_c = \varepsilon$, $t_c = 0$ denote the current node, transition into v_c, and the current duration, respectively, in run r.
8: Initialize $r = run(v_c, v_c, 0)$.
9: **while** $Q \neq \emptyset$ **do**
10: $(t_v, v) = \text{EXTRACT-MIN}(Q)$
11: Create $e = \langle v_c, v \rangle$ s.t. $\rho(e) = \langle t_v - t_c, \{l_{v_c} == v\}, \emptyset \rangle$.
 // We assume that l_{v_c} **denotes a unique control variable associated with node** v_c**.**
12: **if** $e_c \neq \varepsilon$ **then**
13: $(t', g', a') = \rho(e_c)$.
14: $\rho(e_c) = \langle t', g', \{l_{v_c} = v\} \rangle$
15: **end if**
16: $r = r \cdot run(v^c, v, t_v - t_c)$, $e_c = e$.
17: $v_c = v$, $t_c = t_v$.
18: **end while**

The following lemma shows that this procedure preserves the demand of runs $r_1, \ldots, r_m$ and is of duration $\Gamma(r_i)$ for any i.

Lemma 2 *$\Delta(r) = \sum_{i=1}^{m} \Delta(r_i)$ and $\Gamma(r) = \Gamma(r_i)$.*

Proof By definition, $\Delta(s)$ for any run s is equal to the total execution requirement of tasks released by all the nodes in the run. Since Procedure 3 adds all nodes in each run r_i to the run r, the first result follows.

Consider any node v belonging to run r_i. We show that the duration of the partial run leading to node v in run r is the same as that in run r_i. In Line 11 of the procedure, we set the jitter of the transition to $t_v - t_c$ where t_c is the duration of r leading to node v_c. But, since v_c precedes v in run r, the duration of partial run of r leading to v is $t_c + (t_v - t_c) = t_v$. This is equal to the duration of partial run leading to v in r_i. Since this holds for all nodes in r, we get $\Delta(r) = \Delta(r_i)$. □

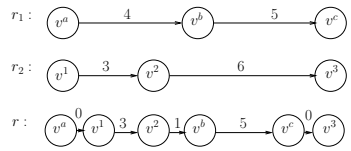


Figure 7. Example for MERGE

The abstraction generation technique uses MERGE to generate the RTC abstraction using a procedure similar to Algorithm 2. Due to space constraints, we do not describe this technique, but only point out the differences from Algorithm 2. (1) RTB-ISO-GEN is replaced by RTC-ISO-GEN, (2) For every critical time instant t (Lines 2,10,11), dbf(t) for the abstraction is generated using MERGE, instead of adding the demands from underlying critical runs. Soundness of the RTC abstraction w.r.t scheduling feasibility is similar to the RBT case.

Properties of RTC abstraction. (1) Ψ is isochronous, but does not satisfy reset frame separation property. (2) The size of abstraction Ψ is $O(\sum_{i=1}^{m} |E_i| S^2 (\log P + \log S))$ where $S = \sum_{j=1}^{m} |V_j|$. This can be explained as follows. The total number of nodes, new control variables, and number of inserted resets in Ψ are all $O(S)$. The size of each transition is $O(\log P + \log S)$ since $O(\log S)$ space is required for storing the new enabling conditions and assignments. The total number of transitions in Ψ is of the order $O(\sum_{i=1}^{m} |E_i| S^2 (\log P + \log S))$. This can be observed by noting that every transition of an underlying RTC model Ψ_i can be duplicated at most S^2 times. Therefore, the total size of Ψ is $O(\sum_{i=1}^{m} |E_i| S^2 (\log P + \log S))$.

6 Compositional Analysis

As mentioned in the introduction, in compositional analysis of hierarchical frameworks, system-level schedulability analysis is done by combining interfaces that abstract component-level timing requirements. In this section, we discuss compositional analysis of conditional task models using our abstractions. Given reset frame separated RBT models, the RBT abstraction generated in Section 4 is also reset frame separated. Therefore, given a hierarchical framework in which all the components are comprised of

RBTs, we can analyze scheduling feasibility using our technique. However, if the underlying RBT models are not reset frame separated, then RTB-ISO-GEN the modified abstraction technique using rbf_Ω instead of dbf_Ω can be used. Intuitively, since rbf accounts for demand of all the task releases in a time interval, it compensates for the loss of demand resulting from the transformation. In general, this modification can result in larger demand overhead and hence, it is beneficial to have reset frame separation.

Similarly, given reset frame separated RTC models, a RTC abstraction can be generated. as in Section 5. However, this abstraction does not satisfy reset frame separation property. A modification to RTC-ISO-GEN, similar to the aforementioned one, can be used to overcome the problem. We can then perform compositional analysis of components comprised of RTC task models.

The RBT abstraction algorithm only uses the demand of the underlying models. Hence, it can be used to generate an RBT abstraction for RTC models without modification. Finally, by observing that an RBT model is trivially an RTC model with no variables and has constant simple runs property, we can compositionally analyze a system comprising of both RBT and RTC models.

7 Conclusions

In this paper, we have introduced RBT/RTC models and techniques for their compositional analysis. This enables modeling and analysis of many real-time applications with hierarchical scheduling policies and conditional real-time code. Although this paper focused on scheduling feasibility, similar techniques can be used for checking schedulability with fixed priority schedulers such as RM. We now present comparisons between RBT and RTC abstractions in terms of size and demand overhead. We note that both the abstraction algorithms convert the underlying models to isochronous models, each having the same period. Therefore, in this discussion, we assume that all the input models have period P.

Size. Traditionally, viewing RBT models as transition systems, their concurrent execution can be represented using a cross-product. To construct a cross-product, we have to assume a synchronization between the models. For models $\Omega_1, \ldots, \Omega_m$ and a fixed synchronization between them, the total number of nodes and transitions in the cross product are $O(\prod_{i=1}^{m} |V_i|)$ and $O(\sum_{i=1}^{m} (\prod_{j=1}^{m} |V_j| \log P))$, respectively. This results in a total size of $O(m \prod_{j=1}^{m} (|V_j| \log P))$. However, for schedulability analysis, we must consider all possible synchronizations between the models which implies $O(\prod_{j=1}^{m} |V_j|)$ number of initial nodes. Representing each such synchronization as a product would therefore result in a total size of $O(m \prod_{j=1}^{m} (|V_j|)^2 \log P)$. In comparison, our RBT abstraction has a total size of $O(P \log P)$.

Given RTC models $\Psi_1, \ldots, \Psi_m$, using similar arguments, the size of a cross product that preserves the demand of underlying models is $O(\sum_{i=1}^{m} |E_i||V_i|(\prod_{j \neq i} |V_j|^2) \log P)$. The RTC abstraction, in comparison, has a total size of $O(\sum_{i=1}^{m} |E_i| S^2 (\log P + \log S))$ where $S = \sum_{j=1}^{m} |V_j|$. We observe that this abstraction is exponentially smaller than the product. Additionally, $|E_i| = O(|V_i|)$ when all Ψ_i are RBTs, and the RTC abstraction is at least as big as the corresponding RBT abstraction if and only if $S^3 = \Omega(P)$.

Demand overhead. Unlike RBT abstractions, RTC abstractions may not be reset frame separated. Hence, RTC abstractions may have additional demand overhead in comparison to RBT abstractions. This is because procedure RTC-ISO-GEN now uses rbf instead of dbf.

References

[1] 3ts system. `http://htl.cs.uni-salzburg.at/examples.html`.

[2] M. Anand, A. Easwaran, S. Fischmeister, and I. Lee. Compositional analysis of conditional real-time task models. Technical report, University of Pennsylvania, MS–CIS–07–20, 2007.

[3] S. K. Baruah. Feasibility analysis of recurring branching tasks. In *ECRTS*, pages 138–145, 1998.

[4] S. K. Baruah. A general model for recurring real-time tasks. In *RTSS*, pages 114–122, 1998.

[5] U. D. Bordoloi and S. Chakraborty. Interactive schedulability analysis. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 147–156, Washington, DC, USA, 2006. IEEE Computer Society.

[6] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proc. of IEEE Real-Time Systems Symposium*, pages 26–35, December 2002.

[7] A. Ghosal, A. Sangiovanni-Vincentelli, C. M. Kirsch, T. A. Henzinger, and D. Iercan. A hierarchical coordination language for interacting real-time tasks. In *EMSOFT '06*, pages 132–141, 2006.

[8] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proc. of Euromicro Conference on Real-Time Systems*, July 2003.

[9] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proc. of IEEE Real-Time Systems Symposium*, pages 99–110, December 2005.

[10] J. Regehr and J. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proc. of IEEE Real-Time Systems Symposium*, pages 3–14, 2001.

[11] S. Saewong, R. Rajkumar, J. Lehoczky, and M. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proc. of Euromicro Conference on Real-Time Systems*, June 2002.

[12] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of IEEE Real-Time Systems Symposium*, pages 2–13, December 2003.

[13] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proc. of IEEE Real-Time Systems Symposium*, December 2004.

[14] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM International Conference on Embedded Software (EMSOFT '06)*, pages 34–43, October 2006.