8-2013

A Comparison of Compositional Schedulability Analysis Techniques for Hierarchical Real-Time Systems

Madhukar Anand
Cisco Systems

Sebastian Fischmeister
University of Waterloo

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

A Comparison of Compositional Schedulability Analysis Techniques for Hierarchical Real-Time Systems

Abstract
Schedulability analysis of hierarchical real-time embedded systems involves defining interfaces that represent the underlying system faithfully and then compositionally analyzing those interfaces. Whereas commonly used abstractions, such as periodic and sporadic tasks and their interfaces, are simple and well studied, results for more complex and expressive abstractions and interfaces based on task graphs and automata are limited. One contributory factor may be the hardness of compositional schedulability analysis with task graphs and automata. Recently, conditional task models, such as the recurring branching task model, have been introduced with the goal of reaching a middle ground in the tradeoff between expressivity and ease of analysis. Consequently, techniques for compositional analysis with conditional models have also been proposed, and each offer different advantages. In this work, we revisit those techniques, compare their advantages using an automotive case study, and identify limitations that would need to be addressed before adopting these techniques for use with real-world problems.

Keywords
real-time and embedded systems, compositionality, state-based scheduling, real-time systems

A Comparison of Compositional Schedulability Analysis Techniques for Hierarchical Real-time Systems

MADHUKAR ANAND, Cisco Systems
SEBASTIAN FISCHMEISTER, University of Waterloo
INSUP LEE, University of Pennsylvania

Schedulability analysis of hierarchical real-time embedded systems involves defining interfaces that represent the underlying system faithfully and then compositionally analyzing those interfaces. Whereas commonly used abstractions, such as periodic and sporadic tasks and their interfaces, are simple and well studied, results for more complex and expressive abstractions and interfaces based on task graphs and automata are limited. One contributory factor may be the hardness of compositional schedulability analysis with task graphs and automata. Recently, conditional task models, such as the recurring branching task model, have been introduced with the goal of reaching a middle ground in the tradeoff between expressivity and ease of analysis. Consequently, techniques for compositional analysis with conditional models have also been proposed, and each offer different advantages. In this work, we revisit those techniques, compare their advantages using an automotive case study, and identify limitations that would need to be addressed before adopting these techniques for use with real-world problems.

Categories and Subject Descriptors: C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

General Terms: Theory, Design, Performance

Additional Key Words and Phrases: compositionality, state-based scheduling, real-time systems

ACM Reference Format:
Anand, M., Fischmeister, S., and Lee, I. 2011. A Comparison of Compositional Schedulability Analysis Techniques for Hierarchical Real-time Systems. ACM Trans. Embedd. Comput. Syst. 9, 4, Article 39 (March 2010), 33 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The increasing complexity of real-time embedded systems should be met with equally sophisticated design methods. Component-based analysis techniques promise to deliver such methods, because they approach modeling, designing, and realizing such systems with a divide-and-conquer strategy. This strategy also applies to the temporal domain, where real-time systems must meet temporal requirements that are often expressed as deadlines. A schedulability check decides whether or not the system can meet these deadlines. In this context, component-based approaches naturally lead to hierarchical scheduling frameworks with hierarchical resource sharing under different policies. To take advantage of a 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 combine component interfaces that abstract component-level timing requirements.

Checking schedulability in a hierarchical real-time system consists of several steps, the first of which is to choose interfaces that are sufficiently expressive to represent components, only exposing

This work is supported in part by NSF CNS-0834524, ARO W911NF-11-1-0403, NSERC DG 357121-2008, ORF RE03-045, ORE RE-04-036, and ISOP IS09-06-037.
Author's addresses: M. Anand, Cisco Systems, Inc. San Jose, USA; S. Fischmeister, Department of Electrical and Computer Engineering, University of Waterloo, Canada; I. Lee, Department of Computer and Information Science, University of Pennsylvania, USA.

information that is needed for the latter analysis. Real-time systems have traditionally been represented using task frameworks such as periodic/sporadic tasks, task graphs, and timed automata. Schedulability is a well studied problem for hierarchic periodic/sporadic representations, but results for more sophisticated representations such as task graphs and timed automata are limited. One contributory factor may be the hardness of compositional schedulability analysis with task graphs and automata. A subclass of task graph models, called "Recurring Task with Branching" (RTB) model and "Recurring Branching Task Model with Control Variables" (RTC), hereafter collectively referred to as *conditional task models*, have been introduced as generalizations of periodic, sporadic, and multi-frame tasks. These models represent real-world applications more accurately than periodic and sporadic tasks, because they can capture the branching behavior and dependencies found in real-world applications. In our earlier work [Anand et al. 2008], we developed efficient schedulability and compositional analysis techniques for these models. This subclass of the task graph model therefore represents a middle ground in the tradeoff between expressivity and hardness of schedulability analysis, and is the focus of our work.

The second aspect in the analysis of hierarchical embedded real-time systems is solving the problem of interface composition. In recent years, increasing attention has been given to the compositional analysis of hierarchical scheduling frameworks. Traditionally, this analysis has been performed 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 [Feng and Mok 2002], and Shin and Lee [Shin and Lee 2004] addressed the interface generation problem for these models. Similarly, studies [Saewong et al. 2002; Lipari and Bini 2003; Shin and Lee 2003] have been carried out on the component abstraction problem using periodic resource models. When these techniques are used for hierarchical frameworks with conditional task models, the complexity of the interface generation depends on the utilization of the task set (i.e., the schedulability checking of conditional task models depends on the utilization [Baruah 1998a]). Specifically, if the utilization is high, then these procedures will be inefficient. On the other hand, compositional analysis techniques for recurring branching models (see [Anand et al. 2008]) synthesize component interfaces from the resource demand of conditional task models in those components. These techniques are independent of task set utilization, and therefore, do not suffer from the same drawback as resource-model based techniques; however, they do so at the expense of increasing the system complexity.

It must be noted that compositional schedulability analysis can be done for every type of task model, including periodic, RTB/RTC, and task graphs. Depending on the choice of the task model, the complexity computing the task demand of a composite component will be different. Related to this complexity is the question of how precise (close enough to the aggregate demand of underlying tasks) the abstract task demand is. In this work, we mostly focus on the second question, and show that RTB/RTC task models offer a good trade-off. The abstract RTB/RTC model captures underlying task demand more closely than periodic task model, but are simpler to construct and analyze than task graphs.

Methodology and Overview. In this paper, we work with new models of real-time tasks that are strictly more expressive than traditional real-time task models, such as the periodic and multiframe models, as well as the recurring task model as proposed by Baruah [Baruah 1998a], but at the same time these new models retain efficient demand computation. Specifically, we support analysis for models where the period of recurrence of different branches is not identical (anisochronous) and for models with explicit control variables, transition guards, and assignments. We call these models the RTB and RTC models, respectively.

Using the RTB and RTC, we focus on the analysis of hierarchical scheduling frameworks. We believe that to be faithful to the paradigm of component-based development, it is desirable to carry out schedulability analysis of a hierarchical systems in a compositional manner, i.e., a system-level schedulability analysis should combine component interfaces that abstract component-level timing requirements. We work with techniques to abstract the resource demand of RTB/RTC models into interfaces (that are in turn RTB and/or RTC models), and thus address the compositional analysis for these models.

Contribution of this work. In this work, we apply compositional analysis techniques to the study of the schedulability of an automotive control application with the aim of gauging the effectiveness of the abstraction techniques and weighing the tradeoffs with respect to system analyzability, abstraction overhead, ease of modeling, etc. involved in a real-world application. Specifically, we have four goals for this study:

(1) Evaluate RTC in a real-world application,
(2) study the impact of task models on the schedulability analysis of the application,
(3) illustrate compositional analysis for hierarchical scheduling frameworks, and
(4) study the impact of the choice of abstractions for the compositional schedulability analysis.

In the case study, we look into the requirements/considerations for a Class C vehicle communication system as described by the Society for Automotive Engineers (SAE) in their report SAE $J2056/1$ [SAE 1993]. The Class C vehicle communication specification models a multiplex system marked by high-data communication rates and communication to support real-time control systems such as engine control and anti-lock brakes. In the multiplex setup, components communicate over a shared signal bus, as opposed to point-to-point links, to facilitate distributed control and to reduce vehicle wiring. The shared bus raises concerns about access contention, which must be resolved using an appropriate scheduling algorithm in a way such that all the timing requirements (latencies) of all the subsystems and messages are met.

Several bus technologies, such as the Controller Area Network (CAN) [Tindell and Burns 1997] and the Time Triggered Protocol (TTP) [Kopetz 1994], provide the means to implement the SAE class C application. Each of these technologies provides a different approach to bus arbitration. CAN takes a dynamic priority-based approach, whereas TTP uses a predefined time-triggered schedule. In contrast to these specific arbitration techniques, our objective here is not to describe a scheduling algorithm, but rather to study the impact of task models and abstractions in analyzing the schedulability of the application when using a hierarchical framework. We therefore do not consider attributes such as fault tolerance, message optimizations, or any other overhead associated with a particular choice of bus technology as all of these can be incorporated into the analysis when desired.

The rest of the paper is organized as follows. Section 2 introduces the system, task and resource models. Section 3 presents compositional analysis using resource supply and conditional task models. Section 4 presents highlights of the SAE Class C application communication requirements. Section 5 develops the conditional models for the application and Section 6 develops the abstractions for different modules of the application. Section 7 presents the results of different abstraction techniques for schedulability analysis, and our conclusions are presented in Section 8.

2. MODEL AND DEFINITIONS

In this section, we define the task and supply models along with their 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. *Schedulability* is a property of the tasks and the system resources whereby all the tasks in the system can meet their resource requirements before the stipulated deadline. Schedulability, however, is not concerned with the actual allocation of those resources, which is specified by a scheduling algorithm. In our framework, we assume that the scheduling of tasks is done as per either the Earliest Deadline First (EDF, [Baruah et al. 1990]) or Rate-monotonic (RM [Lehoczky et al. 1989]) algorithms.

A *hierarchical* real-time system comprises one or more components arranged in a scheduling hierarchy, i.e., each component has its own scheduling policy, and the resources are allocated hierarchically from a parent component (This system is assumed to be free of cycles). The shared resource demand of each component is assumed to be represented by a set of Tasks.

A *simple task* $T = (e, d)$ requires e time units of the resource within d time units of its release. Simple tasks express basic resource requirements without attempting to capture any type of interactions or dependencies in the resource requirements of the application. The RTB model uses these simple tasks and captures timing dependencies between simple tasks.

2.1. Sporadic Task Model

A sporadic task $S = (p, e, d)$ has minimum separation p, execution requirement e and relative deadline d such that $e \leq d \leq p$. The resource demand of a component with only sporadic tasks in its workload is the collective resource requirement that its tasks request when they are scheduled using the components scheduling algorithm.

$$\mathrm{dbf}_S(t) = \left\lfloor \frac{t + p - d}{p} \right\rfloor e \tag{1}$$

2.2. Recurring Task with Branching (RTB) Model

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

Definition 2.1 (*RTB Model*). An RTB 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, where $E_R = \{\langle v, v_0 \rangle | v \in V_F\}$ and $E_T = E \setminus E_R$ such that the underlying graph (V, E_T) is a directed tree.
— $\tau : V \to \mathcal{T}$ is a function from nodes to simple tasks.
— $\rho : E \to \mathbb{N}$ is a function from a transition to minimum jitter.

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 RTB model can 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 made after a minimum delay of $\rho(\langle v_0, v \rangle)$, and this process of task release continues from node v. This behavior continues until a leaf node is reached. At the leaf, the execution *resets* and restarts at the initial node v_0.

Example: Consider the example of the Three Tanks System (3TS) [Ghosal et al. 2006] shown in Figure 1. The plant consists of interconnected water tanks, where each tank has evacuation taps for simulating perturbations - Tap1, Tap2, and Tap3. The tanks are interconnected via taps Tap13 and Tap23. The Pumps P1 and P2 can pump water to increase the water level of tanks T1 and T2, whereas the evacuation taps can be used to simulate perturbations in the water level. The goal of the controllers is to maintain the water level in tanks T1 and T2. The plant is nonlinear and therefore uses three different controllers for each pump: (1) A controller P (proportional) is used 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); and (3) when the control error is large, a controller with a fast integration speed is used (otherwise, a controller with a slow integration speed is used).

The controller logic is implemented in a language called Hierarchical Timing Language (HTL). The HTL code specifies the firing (timing) of the controller tasks, actuator tasks, and sensor tasks. The HTL code itself is executed inside a virtual machine (VM) environment that ensures the timing constraints (e.g., the periodicity of HTL tasks) specified by the HTL code are met. The virtual machine in turn executes inside the native operating system. The tasks themselves are written in a high level language such as C, and are executed directly on the native operating system.

The entire system can be modeled using a two-level scheduling hierarchy as shown in Figure 2.2. On the first level of the hierarchy, we have the HTL tasks describing the firing of the controller tasks, the sensor tasks, and the actuator tasks. These HTL tasks are scheduled by the virtual machine (VM) scheduler. These three modules release tasks T_{CT}, T_S and T_A, which perform the control, sensing and actuation. The native operating system is responsible for scheduling these tasks and the VM that runs the HTL tasks. In our example, we assume that the VM uses EDF (SCH_1=EDF) for scheduling the modules and that the operating system uses RM (SCH_2=RM) to schedule the tasks. The firing

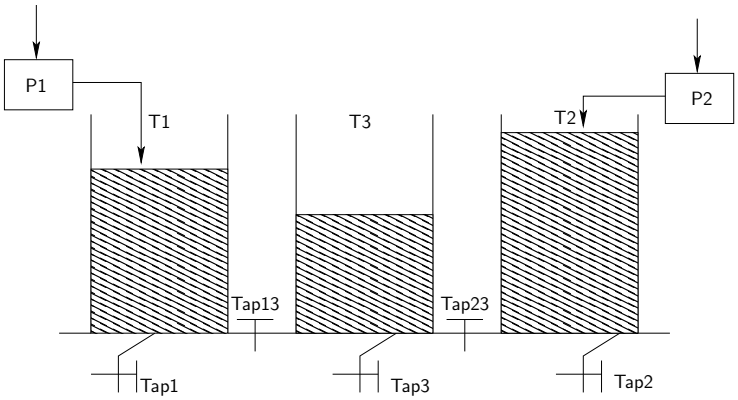


Fig. 1. Overview of 3TS

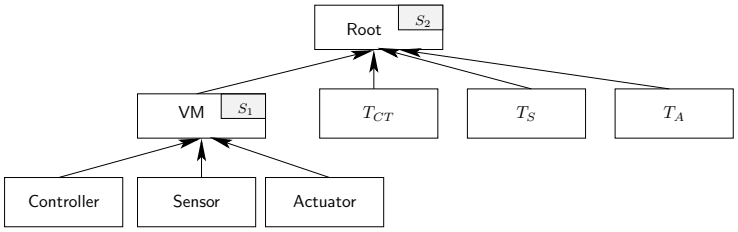


Fig. 2. System hierarchy

logic described in the HTL code is modeled using a conditional task model. Figure 3 shows these conditional models for the three modules of the 3TS.

(1) The controller task (firing) model has two modes of operation. The first mode of operation ($m\_T1\_control\_P$) involves using the proportional controller (P) as mentioned above. The second mode ($m\_T1\_control\_PI$) corresponds to using the proportional integrator (PI) controller. This mode has two sub-modes, which correspond to fast (PI_f) and slow (PI_s) rates of integration. These modes are captured as transitions from the nodes of the RTC model in Fig. 3, i.e., a mode change is modeled as taking a transition to a particular node in the RTC model. In the model, the nodes P, PI_f, and PI_s each release a control task with an appropriate deadline. The leaf nodes $L_1, \ldots, L_6$ do not release any tasks but continue back to the start node. The dangling edges over leaves represent a reset back to the start node. The execution of the controller task starts at node R. Based on the sensor inputs that monitor the water level, one of the controller modes (P/PI) is chosen first. If mode PI is chosen, again, depending on further sensor inputs, the rate of integration is decided. Once the rate of integration is decided, there are different possible transitions to leaf nodes. Specifically, one transition in which the next mode will be set to P, and another in which the leaf node will ensure that the mode remains the same. For example, from node PI_f, there are transitions to L_3, with a guard condition PI_2_P and an assignment of next mode to be P, and a transition to leaf L_4 without changing the mode.
(2) The firing of a sensor is modeled as a conditional model with two modes of operation corresponding to the P or PI modes of the controller. The PI mode, as for the controller task, has two sub-modes corresponding to fast and slow rates of integration. From the node PI, a guard condition $isFast\_PI\_T1$ or $isSlow\_PI\_T1$ (set by the controller) will determine which of the sub-modes is entered.
(3) The actuator firing is modeled as a simple periodic task.

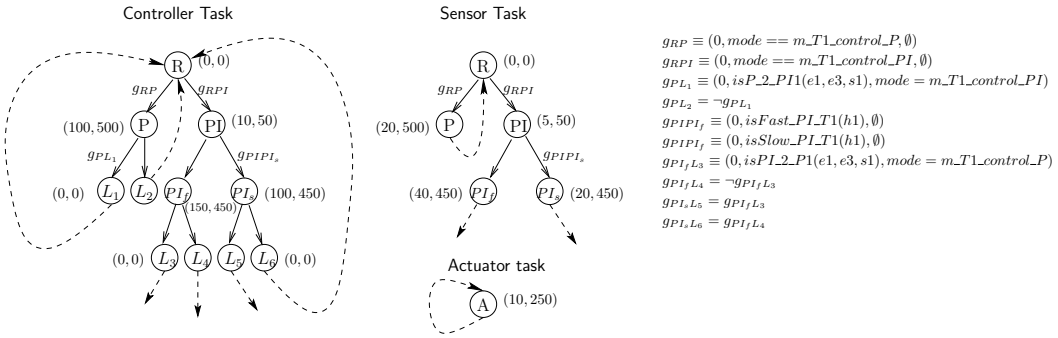


Fig. 3. Conditional task models for the 3TS system

The reader is referred to HTL code [Iercan 2005], which contains the actual programming logic for these modules.

We now introduce some definitions related to the RTB model.

Definition 2.2 (*Run*). For a given RTB model $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$ and a duration t, a run $r \equiv run(v_i, v_{i+j}, t)$ is a a sequence of progressions of nodes from v_i to v_{i+j}. The sequence is defined as $v_i \xrightarrow{e_{i+1}} v_{i+1} \xrightarrow{e_{i+2}} \ldots \xrightarrow{e_{i+j}} v_{i+j}$ 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 the minimum possible duration[1] t of run r by $\Gamma(r)$. 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 $\tau(v_{i+l})$. Also note that the duration parameter t is needed to distinguish runs that involve resets and thus have multiple visits to the root node in the same run.

Definition 2.3 (*Isochronicity*). An RTB model Ω is *isochronous*, if the progression along all branches to the leaves requires the same time: $\forall v_i, v_j \in V_F, \Gamma(run(v_0, v_i, t)) + \rho(v_i, v_0) = \Gamma(run(v_0, v_j, t)) + \rho(v_j, v_0)$. 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 RTB model Ω with period P, we define the worst-case loop as the loop progression with the highest demand starting from v_0, ending at v_0, and passing through exactly one leaf: $wcl(\Omega) = \arg\max_r \Delta(r)$, where $r \equiv run(v_0, v_0, P)$ and $\arg\max_r$ gives the argument at which the function attains its maximum value.

As an example, consider the RTB tasks of the 3TS system shown in Figure 2.2. All the tasks are isochronous. The controller and sensor tasks have a period of 500 time units, whereas the actuator task is a periodic task (trivially an isochronous RTB task) with a period of 250 time units. For the controller task, the cumulative demand posed by different paths (from left to right) are 100 (through L_1 and L_2), 160 (through L_3 and L_4), and 150 (through L_5 and L_6). Therefore, the path from the root through L_3 or L_4 is the worst-case loop of the RTB, with the highest demand being 160 units needed during 500 time units.

2.2.1. The Demand for RTB Models. The resource demand bound function ($\text{dbf}_\Omega : \mathbb{R} \to \mathbb{R}$) of an RTB model Ω defines an upper bound for the amount of resources required to meet all deadlines. For a time interval length t, $\text{dbf}_\Omega(t)$ gives the largest resource demand of Ω for any time interval of length t. This computation is done over tasks that are both released and have their deadlines within the interval. Furthermore, a run r of Ω such that $\Delta(r) = \text{dbf}_\Omega(t)$ is called a *critical run* over the interval length t. In this paper, for ease of presentation, it is assumed that $\Gamma(r) \leq t$. This property

[1]Note that this is actually the minimum possible duration of the run. As we are interested in analyzing the worst case schedulability, we are concerned with this minimum duration.

is known as frame separation.[2] This frame separation is, however, not necessary for the schedulability checking of RTB models (See a similar argument for recurring branching tasks presented by Baruah [Baruah 1998a]). For the purposes of the analysis presented in this paper, it is only required that, for any $v \in V$, the deadline of $\tau(v)$ be at most the duration of the shortest run to v_0 from v, a property known as *reset frame separation*. The request bound function ($\text{rbf}_\Omega : \mathbb{R} \to \mathbb{R}$) of an RTB model Ω, puts an upper bound on 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 the deadlines of which are outside the interval.

For isochronous RTB models, a procedure similar to that proposed by Baruah [Baruah 1998a] can be used to compute the dbf. We summarize this technique below. We assume $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$ 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 $t \geq 2P$, a critical run 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 $t_1 + t_2 + t_3 = t$. Because $run(v_i, v_0, t_1)$ 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 the 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, because the minimal duration between successive invocations of the start node in the anisochronous case depends on the particular run. In fact, it can be easily shown that the problem in NP-hard by reducing the *integer knapsack* problem to the algorithm for computing the dbf for anisochronous models.

We now describe a procedure to transform anisochronous RTB models into isochronous models. This procedure ensures that the demand of the transformed model is at least as high as the demand of the anisochronous model. Consider an anisochronous RTB model $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$. Let $\{r_1, \ldots, r_n\}$ denote the runs from v_0 to each of the leaf nodes v_i, and $\langle \Gamma(r_1), \ldots, \Gamma(r_n) \rangle$ be the set of run durations. Furthermore, without loss of generality, we assume that $\Gamma(r_1) \leq \Gamma(r_2) \leq \ldots \leq \Gamma(r_n)$. For example, consider the anisochronous RTB model shown in Figure 4(a). This model consists of two different runs between successive invocations of the start node, one through leaf v_1 and the second through v_2. In this case, the minimum durations of these runs are $\langle \Gamma(r_1) = 4, \Gamma(r_2) = 7 \rangle$.

We transform Ω to an isochronous model with period P for some $P \geq \Gamma(r_n)$ by adding the dbf_Ω at the end of each run. 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 2 (Appendix A.1). Rather than go into the details here, we illustrate the algorithm using the anisochronous model shown in Figure 4(a) when $P = \Gamma(r_2) = 7$. The resulting isochronous model is shown in Figure 4(b).

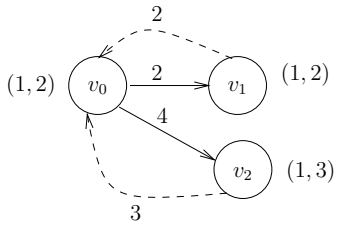
THEOREM 2.4. *(from [Anand et al. 2008]) Let $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$ and P be the input to Algorithm 2, and $\Omega' = \langle V', v_{0'}, V_{F'}, E', \tau', \rho' \rangle$ denote its output. Then, for all $t > 0, \text{dbf}_{\Omega'}(t) \geq \text{dbf}_\Omega(t)$.*

The proof of the above theorem can be found in the dissertation [Anand 2008]. We now present an upper bound on the demand overhead incurred in the conversion technique given by Algorithm 2. If the original anisochronous model is Ω and the corresponding isochronous model is Ω', the *maximal utilization overhead* is defined as follows:

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

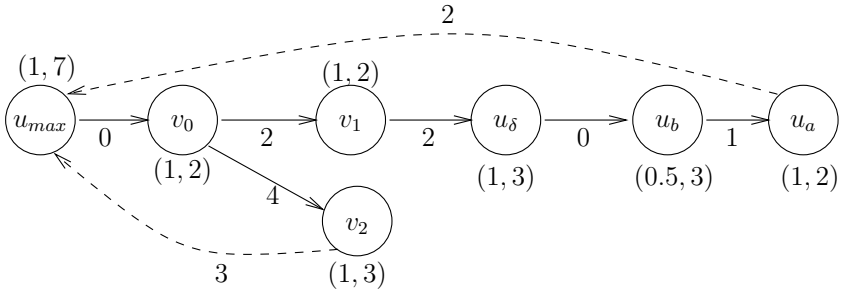
This overhead reflects the increase in utilization that a resource supply would have to support due to the transformation. First, we make the following observation about dbf_Ω in any interval. This observation follows from the reset frame separation property and arguments similar to the dbf computation for isochronous RTB models [Baruah 1998a].

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



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

Fig. 4. Model transformation

PROPOSITION 1. *Given an anisochronous model Ω with the reset frame separation property (with $\langle \Gamma(r_1), \ldots, \Gamma(r_n) \rangle$, representing the minimum durations of runs between successive invocations of the start node through different leaf nodes) and a time interval t, we can partition t into sub-intervals $t_1, \ldots, t_m$, where t_1, $t_m \leq \Gamma(r_n)$, and for $i = 2, \ldots, m-1$ $\exists j$, $1 \leq j \leq n$, with $t_i = \Gamma(r_j)$, such that $\mathrm{dbf}_\Omega(t) \geq \sum_{i=2}^{m-1} \max_{r_i} \Delta_\Omega(r_i) + \max_{r_1, r_m} \Delta(r_1 + r_m)$, where r_i is a run of Ω of duration t_i. 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 $\mathrm{MUO}_{(\Omega,\Omega')}$.

THEOREM 2.5. *(from [Anand et al. 2008][3]) Let $\Omega' = \text{RTB-ISO-GEN}(\Omega, P)$. Then*

$$MUO_{(\Omega,\Omega')} \leq \max_{t < 2\Gamma(r_n)} \frac{\mathrm{dbf}_{\Omega'}(t)}{t} - b_\Omega - \max_{t < 2\Gamma(r_n)} \left\{ \frac{\max_{r \in \mathcal{R}} \Delta(r) - d_\Omega}{t} \right\}$$

where $\mathcal{R}$ is the set of all runs of Ω through v_0 and of duration less than $2\Gamma(r_n)$, r_i is as defined in Proposition 1, and $b_\Omega = \max_i \frac{\Delta(r_i)}{\Gamma(r_i)}$ and $d_\Omega = 2\Gamma(r_n) b_\Omega$.

Note that because Ω' is isochronous, the computation of $\mathrm{dbf}_{\Omega'}$ for values up to $2\Gamma(r_n)$ is straightforward. We refer the interested reader to the dissertation [Anand 2008] for a proof of the theorem.

2.3. Recurring Branching Task with Control variables (RTC) Model

Definition 2.6 (*RTC Model*). The RTC model Ψ is similar to the RTB model Ω (Definition 2.1), except that ρ is now defined as a function from a transition to the minimum jitter, an enabling condition, and a variable assignment.

Formally, $(\rho : E \to \mathbb{N} \times G \times A)$, where $a \in A$ consists of assignments for variables in $\mathcal{V}$ and $g \in G$ is any decidable function over the variables $\mathcal{V}$. The definitions of a run, Γ, Δ, isochronicity, and *wcl* for RTC models are similar to those for RTB models, except that the transitions in a run must be enabled. Hence, a run is only defined under a fixed variable assignment. In the remainder of the

[3] In [Anand et al. 2008], MUO is referred to as MLO - Maximal Load Overhead. We use slightly different terminology here.

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 and enables this run. The execution semantics of the RTC model are similar to those of the RTB model. In addition, the enabling conditions/variable assignments on a transition from v_i are assumed to be instantaneously evaluated after the release of task $\tau(v_i)$. The right part of Figure 3 shows the 3TS system models with the enabling conditions included.

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

2.3.1. Demand for RTC Models. Given an RTC model Ψ, we denote its demand bound function by dbf_Ψ, and its request bound function by rbf_Ψ. For a recurring real-time task model, Baruah [Baruah 1998b] has presented a technique for computing the dbf with 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 also considers enabling conditions on transitions, can be used for computing the dbf of RTC models. Without any restriction on the RTC model, this procedure has similar properties similar to those of the RTB model.

To make the dbf computation efficient, one way to restrict a RTC model without compromising the expressivity much, is to ensure that any two nodes in the model have at most a constant number of *simple runs* between them. We call this property the *constant-simple-runs* property. A simple run is a run that never resets (i.e., a run that uses only transitions in E_T) and no transition is repeated. The constant-simple-runs property is not overly restrictive, in that it permits multiple simple runs between a pair of nodes where each run is enabled by mutually exclusive constraints. Under this restriction a straightforward extension of the technique for use with RTBs can be used to compute the dbf for RTC models. The algorithm for computing the RTC task abstraction (discussed in Section 3.3.2) solves the computation for the more expressive RTC model with the same run-time complexity as the simpler RTB model. It is our expectation that RTCs with the constant-simple-runs restriction can compactly capture many real-world applications. This expectation stems from the fact that typical real-time applications, such as those involving different modes of operations, allow for a fixed number of different runs. Note that the constant-simple runs restriction is not necessary when modelling using RTCs. Relaxing the restriction will result in an increase in the complexity of the demand computation. For the 3TS example, there exists only one path between any non-root nodes (without considering a reset). The RTC task models in Section 4 also illustrate this clearly for the automotive case study.

For anisochronous RTC models, as with RTBs, we present an algorithm RTC-ISO-GEN in Appendix A.2 to transform the task into an isochronous task.

2.3.2. RTB/RTC in relation to other task modeling frameworks. The periodic task model is the simplest of the task models, followed by sporadic task models with explicit deadlines and multiframe task models, which are generalized from the recurring task models. The RTB/RTC task models generalize these well-known task models to consider cycles of variable length (anisochronous) and guards for transitions. Hence, they are more expressive than the basic task models. However, task-graphs, timed-automata, and process-algebra-based models are, in general, more expressive than RTB/RTC models. However, the additional expressivity of these models comes at the cost of an increased complexity of demand computation. In related work, real-time calculus based approaches (such as [Thiele et al. 2006; Thiele et al. 2000; Chakraborty et al. 2003]), and real-time interfaces ([Matic and Henzinger 2005; Bordoloi and Chakraborty 2006]) focus on schedulability analysis given an input (demand) curve, for example, in the form of a task graph. This approach cannot be directly applied to hierarchical frameworks with conditional task models, where the focus is on synthesizing an abstraction that abstracts the complexities and resource requirements of underlying components (perhaps consisting of several conditional models). For large, complex systems, compositional analysis has the potential to simplify the analysis by reducing its overall complexity, but

this may incur overhead on the demand side. It is possible to use these techniques to synthesize the abstraction, but exploring that is beyond the scope of this paper. This work analyses different compositional analysis techniques, measuring this overhead incurred with different techniques. We refer the interested reader to the dissertation [Anand 2008] for a detailed comparison along these lines.

2.4. Periodic and Bounded-Delay Resource Models

In the next two sections, we summarize resource supply models that we use in this work.

The resource supply for tasks is assumed to be provided according to a resource model (e.g., [Shin and Lee 2003; Lipari and Bini 2003; Feng and Mok 2002]). The *Periodic Resource* model [Shin and Lee 2003; 2004] is a resource model that characterizes a periodic resource allocation to tasks. Such a model $\nu(\Pi,\Theta)$ is a partitioned resource supply such that the Θ allocations of time units every Π time units is guaranteed, where a resource period is a positive integer and a resource allocation time is a real number in $(0,\Pi]$. For a periodic model ν, its supply bound function $\mathrm{sbf}_\nu(t)$ is defined by computing the minimum resource supply for every interval length t as follows (Refer to [Shin and Lee 2003] for the details):

$$\mathrm{sbf}_\nu(t) = \begin{cases} t-(k+1)(\Pi-\Theta) & \text{if } t \in [(k+1)\Pi-2\Theta,(k+1)\Pi-\Theta], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{2}$$

where $k = \max\left(\left\lceil \big(t-(\Pi-\Theta)\big)/\Pi \right\rceil, 1\right)$.

The *Bounded Delay* model was introduced by Feng and Mok [Feng and Mok 2002]. This resource partition model, denoted as $R_B(\alpha,\Lambda)$, describes the behavior of a partitioned resource that is available at its full capacity at some times, but unavailable at all other times, with reference to a fractional resource $R_F(\alpha)$. The following property holds between $R_B(\alpha,\Lambda)$ and $R_F(\alpha)$: when an event e happens time t after another event e' over R_F, the time between e and e' over R_B is between $t-\Lambda$ and $t+\Lambda$ (Refer to [Feng and Mok 2002] for details).

$$\mathrm{sbf}_{R_B}(t) = \begin{cases} \alpha(t-\Lambda) & \text{if } t \geq \Lambda, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

2.5. Explicit Deadline Periodic (EDP) Resource Model

The *Explicit Deadline Periodic* (EDP) [Easwaran et al. 2007] resource model provides a periodic resource supply to an application such that the resource is provided before a deadline that is explicitly specified in the model. This choice is implementation-oriented, because many existing real-time schedulers support EDP-model semantics. This model also characterizes many real-time applications, such as avionics and digital control. This model is more general than the well-known periodic resource model (c.f., [Shin and Lee 2003; Lipari and Bini 2003]), in that the deadlines for EDP models are different from their periods. Specifically, an EDP resource model $\xi = (\Pi,\Theta,\Lambda)$ periodically provides Θ units of resource within Λ time units, where the period is Π.

We now compute the resource supply using an EDP model. An EDP model, $\xi = (\Pi,\Theta,\Lambda)$, is a resource model, in which Θ units of resource supply will be provided within $\Lambda(\leq \Pi)$ time units, and this process will be repeated every Π units. We define the bandwidth of this model as $\frac{\Theta}{\Pi}$. Note that a periodic resource model $\nu = (\Pi,\Theta)$ is equivalent to the EDP model (Π,Θ,Π). The supply bound function for this model, sbf_ξ is given in another work (Refer to [Easwaran et al. 2007] for the details).

$$\mathrm{sbf}_\xi(t) = \begin{cases} \left\lfloor \frac{t-(\Lambda-\Theta)}{\Pi} \right\rfloor \Theta + \left(t-(\Pi+\Lambda-2\Theta) - \left\lfloor \frac{t-(\Lambda-\Theta)}{\Pi} \right\rfloor \Pi\right)_0 & t \geq \Lambda-\Theta \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

3. COMPOSITIONAL ANALYSIS

In previous sections, we introduced the conditional task model and described techniques to compute the resource demand in the conditional task model. In this section, we discuss the compositional schedulability analysis for a hierarchical resource sharing system, such as the one described in Figure 2.2, in which the components comprises tasks modeled as RTB or RTCs.

Resource-model-based component interfaces and their compositional analysis are well known (e.g., [Lipari and Bini 2003; Saewong et al. 2002; Shin and Lee 2003; 2004]). Resource models represent the characteristics of resource supplies, and have been used extensively for schedulability analysis. The idea here is to first synthesize a resource supply that meets the scheduling demands of all the tasks at any particular level of the system hierarchy. The next step involves generating a task abstraction at the next level of the hierarchy that guarantees the resource supply underneath. For example, consider a system with hierarchical resource sharing as illustrated in Figure 5. At the bottom of the hierarchy, there are components C_1, C_2, and C_3 with individual scheduling policies. For C_1 and C_2, scheduling is done as per EDF, and for C_3, a rate-monotonic scheduler is used. The component C_4 comprises sub-components C_1 and C_2 and has its own scheduling policy. The compositional schedulability analysis of the system is carried out in two steps. In the first step, a resource supply SCH_{12} is synthesized in such a way that it can meet the resource requirement of tasks T_1 and T_2 of component C_1. In the next step, a task T_{12} is synthesized, so that resources allocated to that task can ensure a supply of SCH_{12} for its underlying tasks. Similarly, a supply of SCH_{34} is synthesized for tasks T_3 and T_4, and this supply is transformed into the task T_{34}. At the level of component C_4, checking for schedulability involves ensuring that tasks T_{12} and T_{34} meet their requirements under the local scheduling policy (EDF in this case).

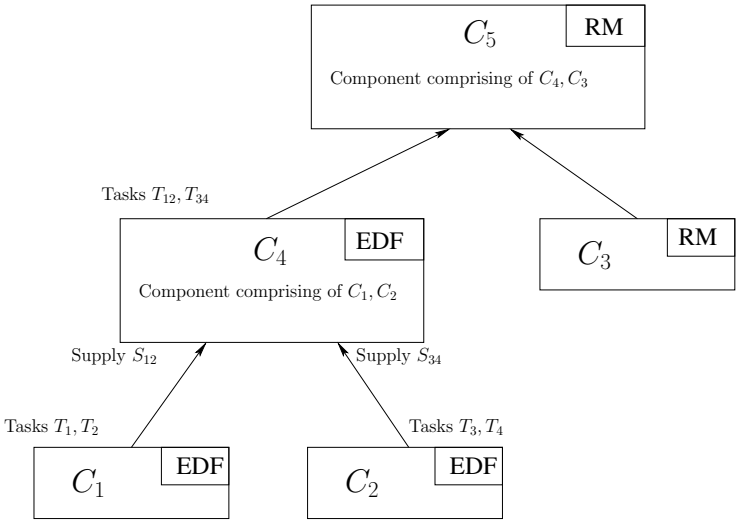


Fig. 5. Hierarchical Scheduling and Compositional analysis

Returning to the 3TS example, as per the system hierarchy in Figure 2.2, a compositional analysis of this situation would involve synthesizing a supply for the controller, sensor, and actuator tasks under the scheduling policy SCH_1 of the VM. Once this supply is has been synthesized, the second step would be to convert the supply to a task at the next level (say T_{VM}) and the schedulability analysis performed with tasks T_{VM}, T_{CT}, T_S and T_A with respect to the scheduling policy SCH_2.

As mentioned in the introduction, in the compositional analysis of hierarchical frameworks, system-level schedulability analysis is carried out by analyzing the combined resource requirements of components via interfaces that abstract the component's resource requirements. Compositional analysis using resource models involves two steps. In the first step, given the component tasks and

their demands, a resource supply Ω is computed for a component C such that the supply always meets the demand. For instance, for component feasibility, $\forall t, \text{sbf}_\Omega(t) \geq \sum_{\tau_C} \text{dbf}_\Omega(t)$ (P_1). Additionally, for optimal abstraction, $\exists t, \text{sbf}_\Omega(t) = \sum_{\tau_C} \text{dbf}_\Omega(t)$ is ensured. In the second step, we map the resource model onto a task at the next level of the hierarchy. The task mapping is such that the abstraction task τ_Ω realizes for the next level that $\forall t, \text{dbf}_{\tau_\Omega}(t) \geq \text{sbf}_\Omega(t)$ (P_2).

For the example in Figure 5, let T_1 be a simple periodic task $(25,4)$, i.e., for every 25 time units, the task requires 4 units of the resource. If T_2 is a simple periodic task $(40,5)$, then a periodic resource supply of $\nu = (10,3.1)$ will meet the demand (based on the first criteria of P_1). With this supply, a periodic task $(10,3.1)$ at the next level meets the criteria P_2, and thus serves as an abstraction for its underlying tasks. Note that this abstraction is not optimal ($\nexists t, \text{sbf}_\Omega(t) = \sum_{\tau_C} \text{dbf}_\Omega(t)$), because it adds demand overhead (resource demand that is strictly more than what is required for underlying tasks) through the process of abstracting the resource requirements.

Several techniques have been proposed for compositional schedulability analysis using resource models under both fixed-priority [Almeida and Pedreiras 2004; Davis and Burns 2005; Lipari and Bini 2003; Saewong et al. 2002] and EDF [Shin and Lee 2003]. For our case study, we use the periodic resource model-based techniques proposed by Shin and Lee [Shin and Lee 2003] and the EDP based techniques proposed by Easwaran et al [Easwaran et al. 2007].

3.1. Compositional Analysis using Periodic Resource Model

A periodic resource model is specified by $\nu = (\Pi, \Theta)$, where Π is the period and Θ is the amount of the resource provided. The supply of a periodic resource model is given by,

$$\text{sbf}_\nu(t) = \begin{cases} y\Theta + \max\{0, t - x - y\Pi\} & t \geq \Pi - \Theta \\ 0 & \text{Otherwise} \end{cases} \quad (5)$$

In this equation, $x = 2(\Pi - \Theta)$ and $y = \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor$. We use the following two-step procedure to compute an optimal periodic resource abstraction for each component of the SAE application. Again, by optimal we mean that the abstraction incurs the least amount of utilization overhead.

(1) Fix the period of supply Π to the shortest period of any underlying task.
(2) Find the smallest Θ such that (a) $\forall t, \text{sbf}_\nu(t) \geq \text{dbf}_\Psi(t)$, where Ψ is the underlying RTC model and (b) $\exists t, \text{sbf}_\nu(t) = \text{dbf}_\Psi(t)$.

Once we have an optimal ν, we transform ν into an optimal sporadic task τ_ν at the next level of the scheduling hierarchy. This transformation is optimal in the sense that (a) $\forall t, \text{dbf}_{\tau_\nu}(t) \geq \text{sbf}_\nu(t)$ and (b) $\exists t, \text{dbf}_{\tau_\nu}(t) = \text{sbf}_\nu(t)$. The computation procedure for τ_ν is similar to the three-step procedure listed above, with the appropriate changes in dbf and sbf.

3.2. Compositional Analysis using EDP Resource Model

Recall from Section 2.5 that an EDP resource model is specified by $\xi = (\Pi, \Theta, \Lambda)$ where Π is the period, Θ is the amount of resource provided by the deadline Λ. The supply bound function (sbf) for ξ is given by,

$$\text{sbf}_\xi(t) = \begin{cases} y\Theta + \max\{0, t - x - y\Pi\} & t \geq \Lambda - \Theta \\ 0 & \text{Otherwise} \end{cases} \quad (6)$$

where $x = (\Pi + \Lambda - 2\Theta)$ and $y = \left\lfloor \frac{t-(\Lambda-\Theta)}{\Pi} \right\rfloor$. We use the following three step procedure to compute an optimal EDP abstraction for each component of the SAE application. By optimal, we mean that the abstraction incurs the least amount of utilization overhead.

(1) Fix the period of supply Π to the shortest period of any underlying task and use the shortest period of recurrence for sporadic tasks. This step ensures that the task with the shortest period gets the resource in time.
(2) Set $\Lambda = \Theta$ and find the smallest Θ such that (a) $\forall t, \text{sbf}_\xi(t) \geq \text{dbf}_\Psi(t)$, where Ψ is the underlying RTC model and (b) $\exists t, \text{sbf}_\xi(t) = \text{dbf}_\Psi(t)$. Let the Θ meeting these criteria be called Θ_*.

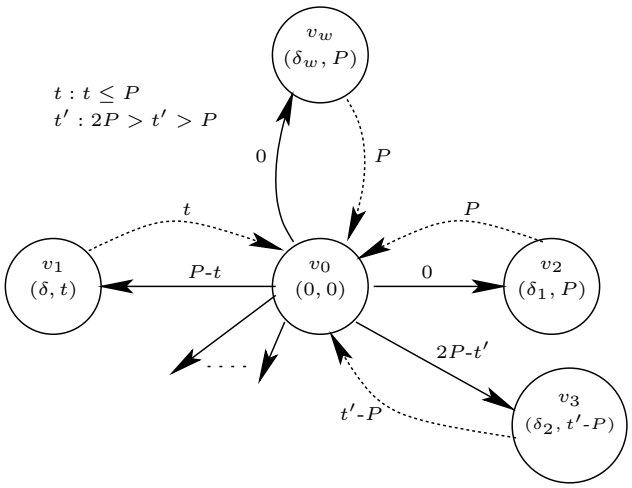


Fig. 6. RTB abstraction

(3) With Π and Θ_*, find the largest Λ such that both criteria (a) and (b) of the previous step are met.

Once we have computed an optimal resource supply ξ for a component, we transform it into a sporadic task τ_ξ for the next level. Again, this transformation is optimal in the sense that (a) $\forall t, \text{dbf}_{\tau_\xi}(t) \geq \text{sbf}_\xi(t)$ and (b) $\exists t, \text{dbf}_{\tau_\xi}(t) = \text{sbf}_\xi(t)$. The computation procedure for τ_ξ is similar to the three-step procedure listed above, with the appropriate changes in dbf and sbf.

3.3. Compositional Analysis using Conditional Models

Compositional analysis based on resource models, although relatively straightforward, have some shortcomings. The first problem with these approaches for compositional analysis is that of bandwidth overhead. The bandwidth of a resource model (e.g., $\frac{\Theta}{\Pi}$ for a periodic resource) is a measure of the resource requirement of the model. It is desirable to minimize this quantity when abstracting component requirements using resource models. For a resource model to be able to schedule a demand, the length of the largest time interval with no supply (henceforth denoted as *starvation length*) must be smaller than the earliest deadline in demand. Because periodic models have implicit deadlines, satisfaction of this requirement depends entirely on its capacity when the resource period is fixed. The overhead incurred by periodic resource models can be significantly reduced by having an explicit deadline for resource supply.

The second problem with previous approaches concerns conditional task models. The resource model based compositional analysis techniques use schedulability conditions to generate component abstractions. For RTB/RTC task models, the complexity of schedulability checking is very high. Schedulability checking for conditional models [Baruah 1998a] is proportional to the average utilization. Therefore, all these approaches for compositional analysis are rendered ineffective for large values of average utilization. To address some of these concerns, abstraction techniques have been developed for a task set comprises recurring task models that does not depend on checking schedulability. These will be our focus for the remainder of this section.

3.3.1. RTB Abstraction. In this section, we describe a technique to abstract a collection of RTB models into one RTB model. Specifically, given models $\Omega_1, \ldots, \Omega_m$, we develop an RTB 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 an RTB abstraction is given in Algorithm 1. First, we transform each model Ω_i into an isochronous model of period $P_m = P$ (Line 1). We add a loop from the start node having a demand equal to the concurrent execution of all $wcl(\Omega_i)$ (Line 2). This loop is shown in Figure 6 as the

loop through node v_w. The dashed line indicates the reset transition. For each $t \leq P$, we add a node with demand $\delta = \sum_{i=1}^{m} \text{dbf}_{\Omega_i}(t)$ (Lines 4-8). This loop is shown in Figure 6 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 6 (Lines 9-14), such that $\delta_1 + \delta_2 = \sum_{i=1}^{m} \text{dbf}_{\Omega_i}(t)$. We observe that the abstraction is isochronous, reset frame separated, and its size is $O(P \log P)$. The following theorem shows that Algorithm 1 generates a sound abstraction with respect to scheduling feasibility.

Algorithm 1 Algorithm for generating RTB abstraction

Input: $\Omega_1, \ldots, \Omega_m$,
where $\Omega_i = \langle V_i, v_0^i, V_F^i, E_i, \tau_i, \rho_i \rangle$.
Input: $P_1 \leq \ldots \leq P_m = P$, where P_i is period of Ω_i.
Input: $C = \{t < 2P | \forall \varepsilon > 0, \exists i, \text{dbf}_{\Omega_i}(t) > \text{dbf}_{\Omega_i}(t-\varepsilon)\}$
Output: $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$, s.t. $\text{dbf}_{\Omega} \geq \sum_{i=1}^{m} \text{dbf}_{\Omega_i}$.
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 $\delta = \sum_{i=1}^{m} \text{dbf}_{\Omega_i}(t)$
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: $\delta_2 = \sum_{i=1}^{m} \text{dbf}_{\Omega_i}(t) - \delta_1$
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**

THEOREM 3.1. *(from [Anand et al. 2008]) If RTB Ω can be feasibly scheduled on a uniprocessor platform, then RTB's $\Omega_1, \ldots, \Omega_m$ can also be feasibly scheduled.*

This result follows from the fact that for all $t > 0$, $\text{dbf}_{\Omega}(t) \geq \sum_{i=1}^{m} \text{dbf}_{\Omega_i}(t)$, which is true by construction for Ω.

Abstraction for 3TS. We apply this abstraction technique to the 3TS example introduced in Section 2.2. For the models shown in Figure 3 using the hierarchical resource sharing framework in Figure 2.2, we ignore the guards and assignments on transitions. Consequently, these models are isochronous RTBs with periods of $P = 500$. The RTB abstraction Ω for the virtual machine scheduling the modules of Figure 2.2 is given in Figure 7.

3.3.2. RTC Abstraction. In this section, we develop an RTC abstraction for a set of RTC models.

The abstraction generation technique uses a procedure MERGE (described in Appendix A.3) that merges critical, concurrent runs to generate the RTC abstraction. The overall procedure is similar to Algorithm 1. Due to space constraints, we do not describe this technique, and instead only highlight the differences from Algorithm 1: (1) RTB-ISO-GEN is replaced by RTC-ISO-GEN and (2) for every critical time instant t (Lines 2,10,11), $\text{dbf}(t)$ for the abstraction is generated using MERGE instead of adding the demands from underlying critical runs. The soundness of the RTC abstraction with respect to scheduling feasibility is similar to the RTB case. For a more detailed description of the algorithm, we refer the interested reader to [Anand 2008].

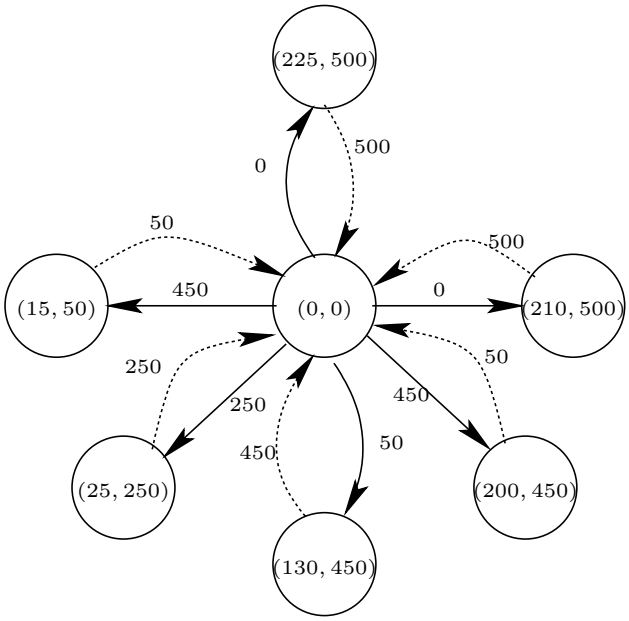


Fig. 7. Abstractions for 3TS

3.3.3. Compositional Analysis with Proposed Abstractions. In this section, we discuss the compositional analysis of conditional task models using our abstractions. Given reset frame separated RTB models, the RTB abstraction generated in Section 3.3.1 is also reset frame separated. Therefore, our technique can be used to analyze the feasibility in a hierarchical system with RTBs as components. If the underlying RTB models are not reset frame separated, then RTB-ISO-GEN should be modified using rbf_Ω instead of dbf_Ω to generate the abstraction. Intuitively, because rbf accounts for the demand of all the task releases in a time interval, the rbf compensates for the loss of demand resulting from the transformation. In general, this modification can result in a larger demand overhead. Therefore, it is beneficial to have reset frame separation.

Similarly, given reset frame separated RTC models, an RTC abstraction can be generated as described in Section 3.3.2. However, this abstraction also does not need to satisfy the reset frame separation property. A modification to RTC-ISO-GEN, similar to that aforementioned, can be used to overcome the problem. We can then perform compositional analysis of components comprises tasks modeled as RTCs.

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

4. SAE CLASS C APPLICATION REQUIREMENT CONSIDERATIONS

In this section, we introduce vehicle communication requirements as listed in the SAE specification document $J2056/1$ [SAE 1993], which we use for our case study. The class C category vehicle communication requirements introduce the aspect of real-time closed-loop feedback in a system. The requirement considerations in $J2056/1$ are clarified with respect to an electrical vehicle drive- and brake-by-wire system. The implementation of the system in an advanced electric vehicle powertrain, called the ETX-I (Electric Trans-Axle), consists of seven modules: the vehicle controller (V/C), inverter motor controller (I/M), instrument panel display, transmission, traction battery, brakes and driver inputs.

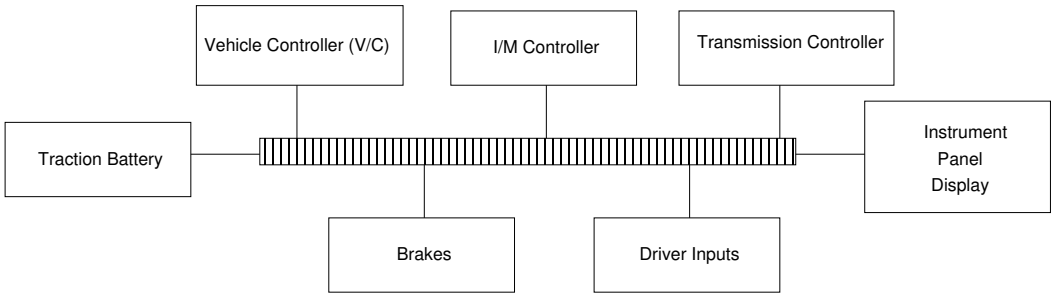


Fig. 8. Block diagram of the different modules

Table I. ETX State diagram definitions

No.	Definition	No.	Definition
0	Initialize vehicle controller	18	Deenergize 1st gear friction clutch
1	Display "NOT IN PARK/NEUTRAL" message	19	Energize 2nd gear friction clutch
2	Close Inverter/Motor controller power relay	20	Deenergize 2nd gear friction clutch
3	Send initialization record to I/M C	21	Start timeout 1 for measured torque
4	Send PTR (Prepare to run) message to I/M C	22	SHUTDOWN
5	Park or Neutral	23	Display diagnostic status
6	Close main contactor negative side	24	Open V/C power relay
7	Close main contactor positive side	25	Send FRV to I/MC
8	Send MCA	26	Send MCR and start timeout for MCC
9	Send NOT(MCR) and start timeout for NOT(MCC)	27	Send NOT(MCR)
10	Open main contactor and start timeout for NOT(MHNA)	28	Set motor overspeed warning flag
11	Send NOT(MCA)	29	Display "CHARGER STILL PLUGGED IN" message
12	Send NOT(FRV) to I/MC	30	Clear motor overspeed warning flag
13	Energize 1st gear friction clutch	31	Decrement Max power available to motor
14	DRIVING	32	Increment Max power available to motor
15	Start time delay 2 for regen to drive shift	33	Increase negative torque to limit motor speed
16	Stop time delay 2	34	Decrease negative torque
17	Start time delay 3 for clutch to drive shift		

The V/C acts as the system's command center. It electronically interprets all driver commands by monitoring the accelerator, brake pedals, and shift lever as well as providing the desired wheel torque response by appropriately controlling the operation of the inverter, motor, transmission, and brake. It also provides fault management and diagnostics. Alternative implementations typically use two dedicated serial links. However, the preferred configuration, as shown in Figure 8, uses a shared bus to enable distributed control and reduce the amount of vehicle wiring resulting in a reduction of the production cost. For further details, the SAE handbook and other references [SAE 1994; Company 1988] contain a detailed description of the propulsion system, its operation, and the design of the control system.

In the following paragraphs, we provide a brief description of the relevant aspects of system operation to illustrate the relationship between system operation and communication requirements. Table I lists all the possible states of the system, and IV list all the control messages that are sent over the communication bus in the ETX-I system.

The initial state of the system is that of the ignition key turned on. Several messages such as the key switch start (KSW, Table IV) and Key switch run (KSR, Table IV) are initiated as a result of this driver action. The V/C needs to know the status of the shift lever, the friction clutches in the transmission, and the relay that locks the power on to the vehicle controller itself. For example, if the transmission is not in "PARK" or "NEUTRAL", then the system will enter the fault state and display an appropriate message. Alternatively, if the transmission is in "PARK" or "NEUTRAL", then the friction clutches will be disengaged and the power relay will be locked on. As the key goes into the "START" position, the system transitions into state numbered 2 in the table, and energizes the relay that provides power to the I/M Controller. The message prepare-to-run (PTR) is signalled

following State 2, after sending the interlock INT, and Power ACK messages to V/C VCA, and the Inverter ICA, respectively. During the time the vehicle moves from "PARK" or "NEUTRAL" to "DRIVING", the V/C sends a Main Contractor Acknowledge (MCA) message to the I/M controller. Thus, the action of the driver in turning the ignition to "ON" and switching the key into the "START" position results in a burst of messages. Although these messages are not recurring, they still should have minimum worst-case latencies to prevent a perceptible delay between key turning and system initialization.

When the system is in State 14 ("DRIVING"), the V/C is repeatedly sending a torque command (e.g., TQC) to the I/M Controller every $5ms$, and at the same time, the V/C needs to receive a pedal position (APP, BPM, BPL, etc.) and calculated torque value (TQM) at the same rate. This is an example of recurring data which must be updated at a rate fast enough to provide a smooth response. If the required time for transmitting the acknowledgment exceeds a particular threshold, then the system may omit the acknowledgment and use the old data until the next data packet arrives. In State 14, the vehicle controller also monitors several other event-driven, single-shot signals which can lead to state transitions (e.g., PBK, SOC, ASW, and BSW). Recipients must acknowledge such messages to prevent faulty state transitions in the system.

One of the ETX-I Control system state transitions will occur, if the driver shifts from 1^{st} to 2^{nd} gear (See State 19 of the table). Because the initiation and execution of the gearshift depend on a time-dependent sequence of certain conditions, it is necessary that the communication system meets the data requirements. Also, while switching gears, the clutch pressure and motor speed signals need to be updated at a much faster rate (5 to $10ms$) than during normal driving in a particular gear. The accelerator switch (ASW), speed control SPC, and shift gear (to park/reverse/drive mode) $SHIFT$ are some of the different messages that are generated during the driving phase, but they are not active concurrently.

The emergency brake message (PBR) and emergency reset (SOC) are two special messages that are get sent by the driver to the vehicle controller in different emergency situations.

The complete list of the messages can be found in the Table IV. In the table, the message type indicates the length of the message in bits. Messages $1-13$, $21, 29, 30, 32$, and 36 of the table are the recurring signals. The other messages are signals that were sent between the vehicle and the inverter/motor controllers. Signals without an update rate associated with them in Table IV are known as event-driven (i.e., driver action or a change in state), and their update rate can only be approximated by a lower bound on successive updates (see Section 5 for a description of how we model them). Some of the signals in the table can cause a state change that might trigger other messages as previously described. Sporadic messages capture event-driven signals.

We now briefly highlight some interesting application characteristics for our schedulability analysis.

— The SAE application very well fits a hierarchical scheduling framework (c.f., Figure 9). The first level of the scheduling hierarchy contains subsystems with specific scheduling policies for these subsystems (e.g., $S_1, \ldots, S_7$). For instance, messages generated by one subsystem can be scheduled inside the subsystem using EDF, whereas others use a FIFO strategy. The second level schedules all messages on the bus, and on this level, the bus arbitration policy decides which messages should be transmitted. For example, in the CAN system, either fixed priority mechanisms [Tindell and Burns 1997] or EDF [Natale and Meschi 2001] policies can be used, whereas in a TTA system, the arbiter uses a strict TDMA policy.
  The SAE system has also been modeled with up to three levels of hierarchy [Meyerowitz et al. 2003], where the extra level of scheduling between the modules performs some optimization.
— The message set consists of both periodic (time-triggered) and sporadic (event-triggered) messages, and the periods of time-triggered messages can differ by several orders of magnitude. That is, the message set contains both messages with very short periods and messages that have long periods. Both of these characteristics affect the abstractions used for compositional analysis. Furthermore, as we will show, the complex inter-message dependencies of the event-triggered part make conditional real-time models more appropriate to model the application than simple periodic or sporadic tasks.

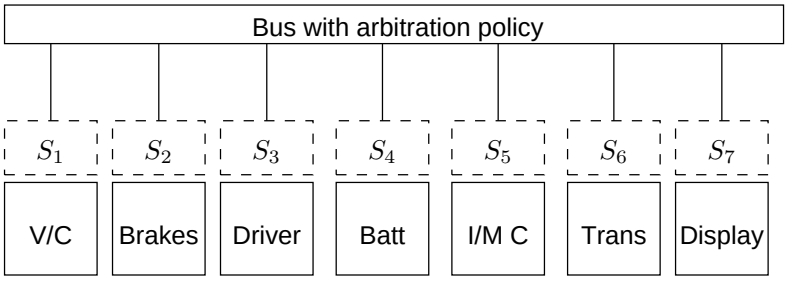


Fig. 9. Scheduling hierarchy for the SAE application

5. MODELING THE SAE APPLICATION

In this section, we model different parts of the SAE application using a two-level hierarchy as indicated in Figure 9. We use mainly the EDF policy for our schedulability analysis, however, fixed priority scheduling can be applied in a similar fashion.

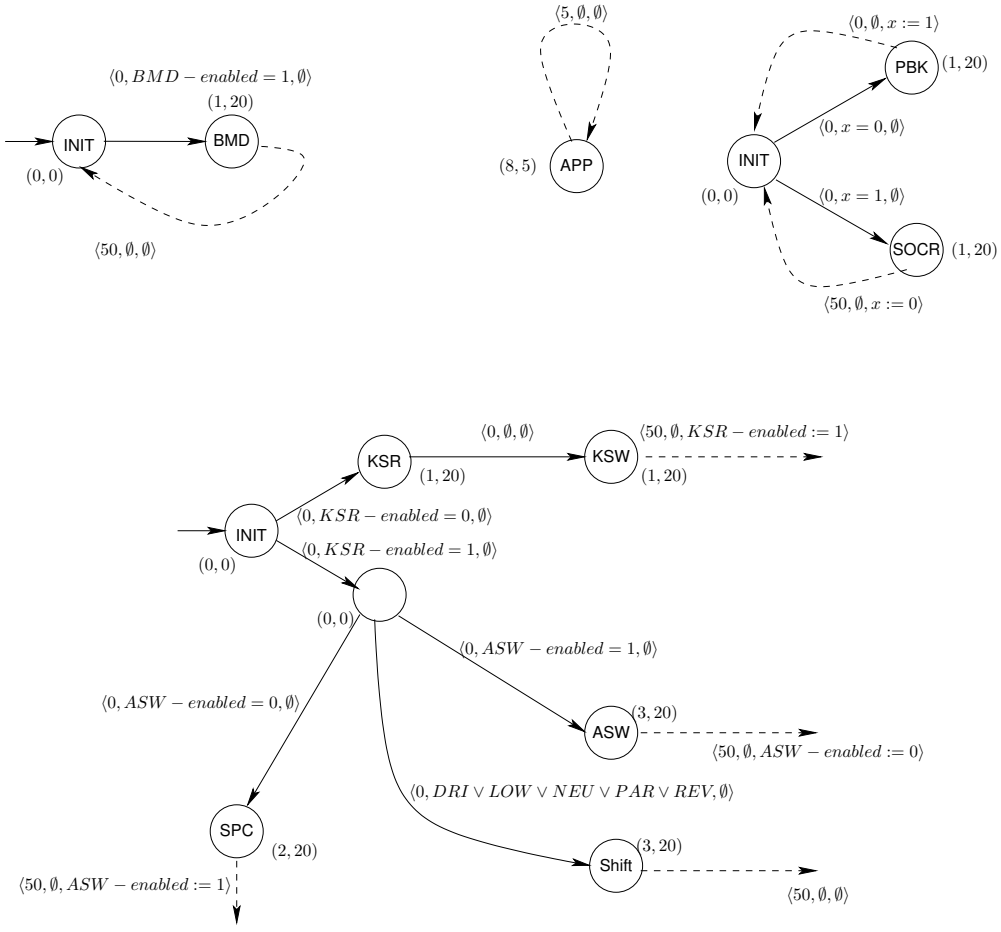


Fig. 10. Task model of the driver module

We begin with the driver module. Table IV indicates that the driver module comprises nine messages, one of which is periodic (*APP*) and eight of which are event-triggered. We use the RTC model for the event-triggered messages, and Figure 10 shows the resulting model. The tuple $\langle c, d \rangle$ on each node denotes the demand of the task c (in bits), and its deadline d (in ms). The tuple $\langle j, g, a \rangle$ denotes the transition's minimum jitter, guard, and assignment. A dashed line specifies a reset to the start location. Below are the different tasks of the driver module.

(1) From Table IV, we see that the driver module sends the brake mode message *BMD*. Correspondingly, we create the first RTC task with a guard variable to determine if the mode is enabled or not.
(2) The message *APP* corresponds to the accelerator position and is sent periodically. The second RTC task for the driver module is, therefore, a simple one node model.
(3) From Section 4, the emergency brake message (*PBR*) and emergency reset (*SOC*) are also exclusive. We model the third task of the driver module considering the exclusivity of these messages.
(4) The remaining event-triggered messages *KSR*, *KSW*, *ASW*, *SPC*, and $SHIFT$ are related as follows. As discussed in Section 4, the Key switch start (*KSW*) and Key switch run (*KSR*) are only active initially. The accelerator switch (*ASW*), speed control *SPC*, and shift gear (to park/reverse/drive mode) $SHIFT$ are not active concurrently. These constraints are captured in the last task of the driver module.

The SAE specification, unfortunately, lacks a lower bound on the inter-arrival rate of event-triggered messages. For our schedulability analysis, we assume a 20*ms* deadline for sporadic messages and a minimum recurrence period of 50*ms*. These parameters are similar to those used in other works [Tindell and Burns 1997; Kopetz 1994]. A closer inspection reveals that using a sporadic model without dependencies for the analysis results in a high overhead. For instance, the messages associated with the ignition system (*KSR* and *KSW*) occur only once in the entire state diagram. Further, the messages never occur simultaneously with the accelerator switch and speed control messages (*ASW* and *SPC* respectively). The RTC model captures such dependencies, e.g., by using the variable KSW-enabled to ensure mutually exclusive messages releases of shift and accelerator messages.

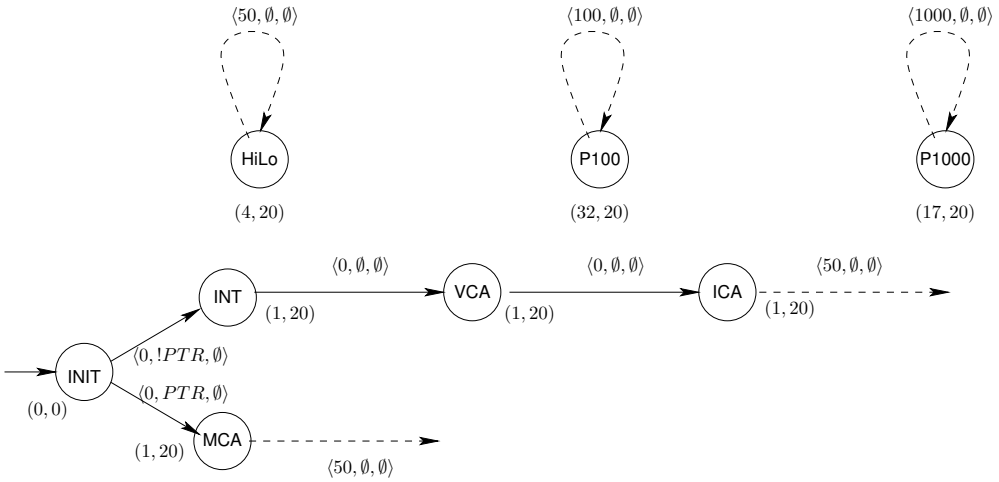


Fig. 11. Task model of the battery module

The battery module comprises twelve messages, of which four recur with a period of 100*ms* and three with a period of 1000*ms*. The remaining five have sporadic behavior. We model the periodic messages as is and represent them as nodes *P*100 and *P*1000, respectively, as shown in Figure 11.

The demand of these nodes is the sum of the individual messages. We model event-triggered messages using the RTC model. This model, again, provides less overhead than the other models. For example, as mentioned in Section 4, the messages *INT*, *VCA*, and *ICA* never occur after the prepare-to-run event *PTR*. We leverage this by defining a variable *PTR* that is true whenever the event *PTR* has happened and false otherwise.

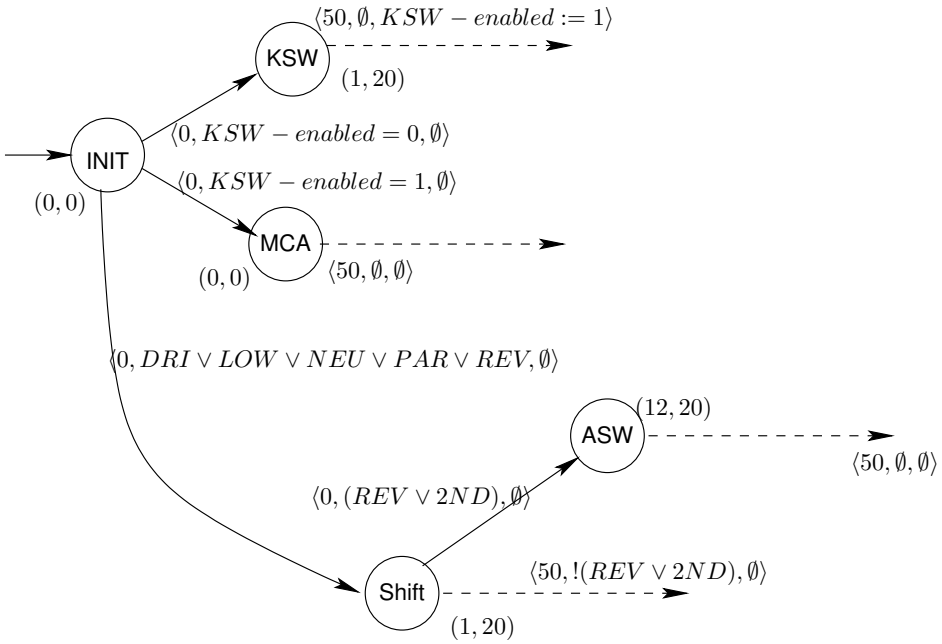


Fig. 12. Conditional task model of the V/C module

The vehicle controller module generates 17 messages. We apply the same methodology to its six periodic messages as shown in the battery model, and we incorporate the remaining eleven messages in an RTC model as shown in Figure 12. The RTC model is based on the observation that the key switch message (*KSW*) occurs initially, followed by the contractor acknowledge message (*MCA*), followed by *SHIFT* and Accelerator switch messages that get triggered depending on the different driving gears.

We then model the remaining modules. The brake module contains only one sporadic message and three periodic messages. The transmission controller generates only periodic messages, and we model these messages as they are. Finally, the I/M controller outputs eight messages, of which six have event-triggered behavior. We model the two periodic messages as they are. Unfortunately, the event-triggered messages are independent of each other, and we therefore model this part using a purely sporadic task model without dependencies.

As a final note, the demand shown in all the figures represents the number of transmitted bits, representing not just the length of the contents but also the required number of bits including, for instance, stuffing bits and CRC data. For the remainder of the paper, unless otherwise specified, we assume that the bus speed is $20kpbs$ when computing the message transfer duration.

6. ABSTRACTIONS FOR DIFFERENT MODULES

We now compare four techniques for computing abstractions of these modules. The choice of techniques is based on the frameworks proposed for the compositional analysis of a hierarchical schedulability framework. We consider the following four techniques:

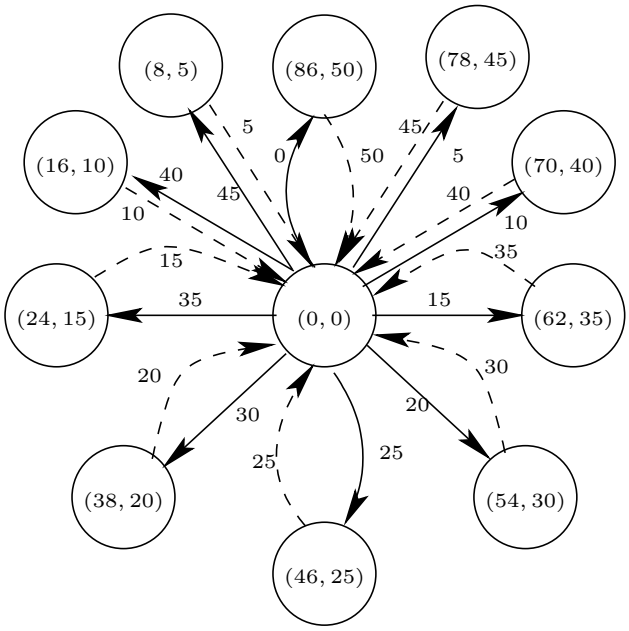


Fig. 13. RTB abstraction of the driver module.

— A conditional task (RTB/RTC) abstraction for the RTC task models as proposed earlier,
— An explicit deadline periodic (EDP) resource abstraction for the RTC task models and subsequent mapping onto a sporadic task,
— An EDP resource abstraction for the sporadic task model and subsequent mapping onto a sporadic task, and finally,
— A periodic resource abstraction for the RTC task models and subsequent mapping onto a sporadic task.

Conditional task abstraction: We use an RTB model for the driver module because its period of recurrence is always at most 50*ms*. Using Algorithm 1, we compute the RTB task shown in Figure 13. The battery module hosts tasks whose period of recurrence varies from 50*ms* to 1000*ms*. An RTB abstraction would result in 40 additional locations to accommodate changes in the demand. We therefore use an RTC abstraction using the technique described in Section 3.3.2 in Figure 14. For the sake of brevity, we omit details for the other modules. However, these can be created in a fashion similar to those presented thus far.

We have highlighted the computation of the optimal EDP abstraction and the corresponding optimal transformation for the Vehicle Controller module in Figure 15. Specifically, the bottom-most function, dbf_Ψ represents the net demand of all the RTB tasks of the controller module. sbf_ξ and dbf_ξ represent the supply function and the corresponding optimal task abstraction of the explicit deadline periodic resource model for the module. sbf_ν and dbf_ν represent the supply function and the corresponding optimal task abstraction of the periodic resource model for the module. The EDP abstraction, as can be seen from the plot, performs better than the optimal periodic resource abstraction. More details on computing the periodic resource abstraction are highlighted below.

In the remainder of this document, to keep the narrative simple, we abuse the notation and denote the sporadic task generated by the EDP resource τ_ξ by ξ.

Figure 15 lists the demand of tasks of the of the vehicle controller module, the demand of the periodic resource abstractions and the demand of the corresponding (optimal) sporadic task abstraction.

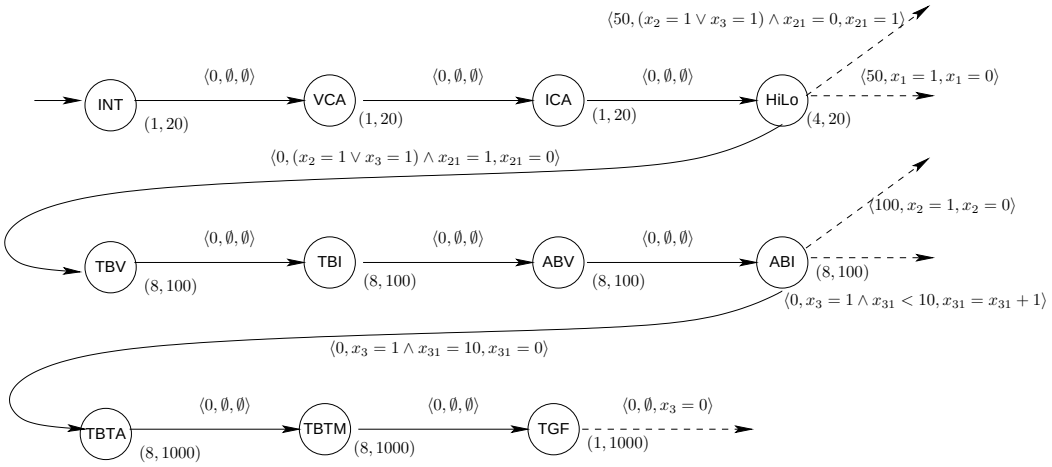


Fig. 14. RTC abstraction of the battery module

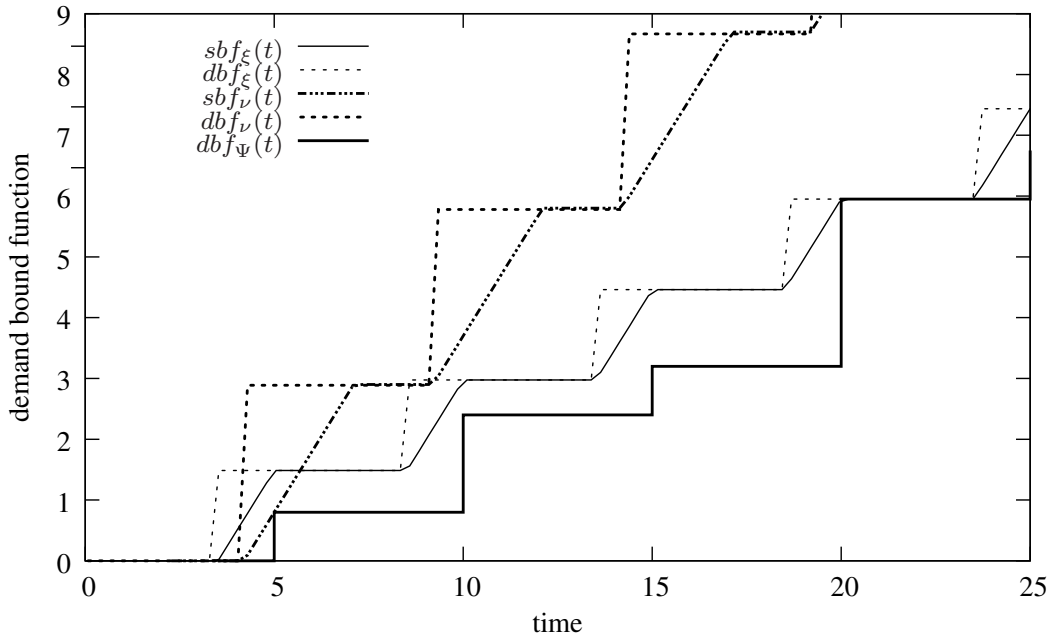


Fig. 15. Computing optimal abstractions (V/C module)

Table II. The table listing EDP and periodic abstractions for different modules

Module	EDP Abstraction Task	EDP Abstraction Task (S)	Periodic Abstraction Task
Driver	$(5, 0.475, 4.525)$	$(5, 0.5625, 4.4375)$	$(5, 2.585, 4.6)$
Battery	$(20, 0.484, 19.516)$	$(20, 0.497, 19.503)$	$(20, 10.175, 19.65)$
V/C	$(5, 1.4875, 3.5125)$	$(5, 1.5125, 3.4875)$	$(5, 2.8895, 4.2)$
I/M C	$(5, 0.975, 4.025)$	–	$(5, 2.8895, 4.2)$
Brakes	$(5, 0.875, 4.175)$	–	$(5, 2.8895, 4.2)$
Trans	$(5, 0.4205, 4.5795)$	–	$(5, 2.6885, 4.6)$

Table II lists the EDP and periodic resource abstractions for all the components of the SAE application. For comparison, we have also listed the optimal EDP abstractions for the case where, instead of conditional models, all the components were modeled using purely sporadic tasks. This is denoted by an (S) in the table. As for the I/M controller, brakes, and transmission modules, the EDP abstractions for sporadic and conditional task cases are identical because we have no explicit conditional models.

7. ANALYSIS AND RESULTS

In this section, we present the results of applying and comparing the abstractions for each of the seven modules of the SAE application. Recall that in Section 2.2.1, we provided a theoretical bound on the maximum utilization overhead for the conditional models. In this section, we use this metric, and the relative utilization overhead to measure the abstraction overhead.

Given a model M and its abstraction A,
Maximum utilization :

$$\mathrm{MU}(M) = \max_{t>0} \frac{\mathrm{dbf}_M(t)}{t}$$

Maximum utilization overhead :

$$\mathrm{MUO}(A,M) = \max_{t>0} \frac{\mathrm{dbf}_A(t)}{t} - \max_{t>0} \frac{\mathrm{dbf}_M(t)}{t}$$

Relative utilization overhead :

$$\mathrm{RUO}(A,M) = \frac{\max_{t>0} \frac{\mathrm{dbf}_A(t)}{t}}{\max_{t>0} \frac{\mathrm{dbf}_M(t)}{t}} - 1$$

The utilization overhead reflects the increase in utilization that a resource supply would have to provide to the application due to the abstraction and an estimate of the demand overhead. The relative utilization overhead describes only how much more demand is incurred by the abstraction relative to the original model.

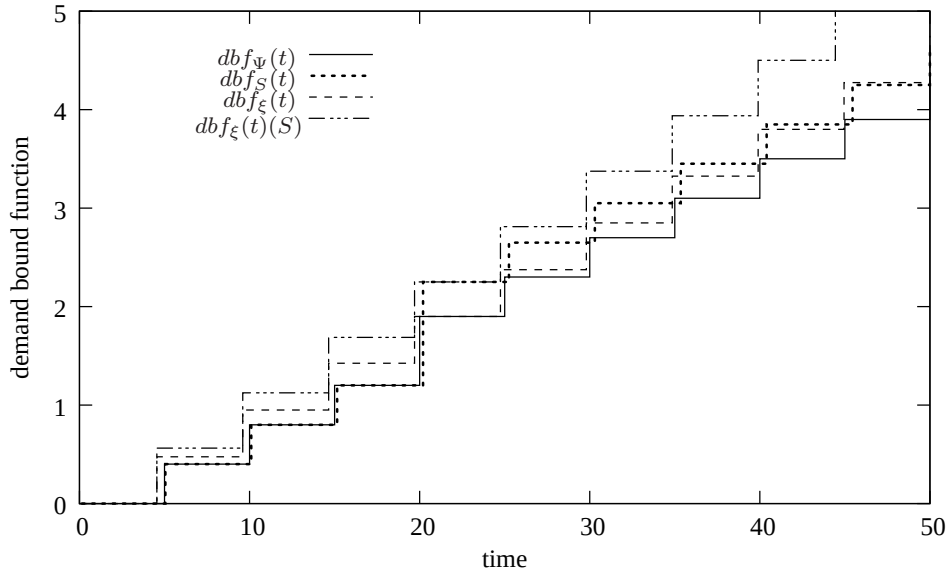


Fig. 16. Demand for the sporadic and conditional task model of the driver module

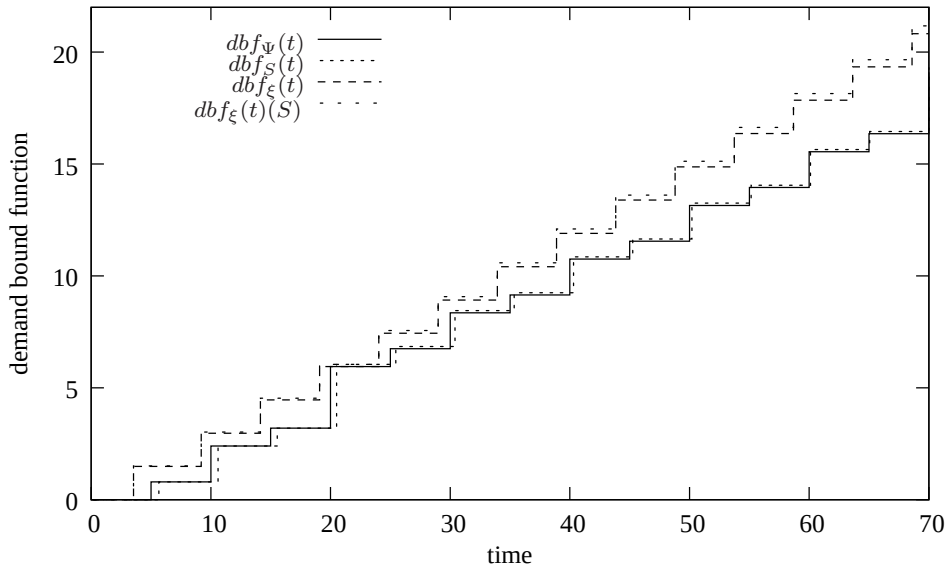


Fig. 17. Demand for the sporadic and conditional task model of the V/C module

Impact of the task model: First, we consider the demand for the conditional task model versus the demand for a purely sporadic model. Figures 16 and 17 plot the demands for the driver and vehicle controller modules. As before, dbf_{Ψ} represents the demand of the module tasks, dbf_{S} is the demand of the sporadic task abstraction, and dbf_{ξ}, and $\text{dbf}_{\xi}(S)$ represent the demand of the task abstraction (at the next level of hierarchy) constructed using an EDP resource model for the underlying conditional and the sporadic task abstractions, respectively. Based on the plots, we can make the following observations:

(1) *The conditional task abstractions incur no demand overhead.* This is because the selected modules have a harmonic recurrence period. Therefore, the transformation algorithms can compute the demand exactly, introducing no additional overhead.
(2) *The conditional task models require less demand.* Based on the figures, it is apparent that the demand of conditional task models is smaller than that of purely sporadic task models. In fact, for the driver module, the maximum utilization is 0.0950 for RTC model (Figure 10), whereas it is 0.1125 for the sporadic task model (an increase of 18.42%). For the battery module, the maximum utilization for the RTC model (Figure 11) is 0.0242 and that of the sporadic task model is 0.0249. The RTC model performs better because it is more expressive than the RTB and does not consider mutually exclusive messages running concurrently. In the case of the battery, the increase is a modest 2.69%, mainly because of a small difference in demands on either of the branches of the RTC model. For the vehicle controller module, these numbers are 0.2975 and 0.3025, a 1.68% increase. The reason for the modest increase is the same as for the battery module.
(3) *A hierarchical abstraction amplifies the benefits of the conditional task model.* Although as illustrated in the previous point, the conditional models have a lower utilization than the sporadic models, the effect of the choice of the task model is more pronounced when they are abstracted. For instance, EDP abstractions for the conditional and sporadic task models for the driver module incur utilization overheads of 0.0100 and 0.0318, an increase in utilization of 10.5% and 33.43%, respectively. For the battery module, because the difference in utilization between RTC and sporadic task models is small, the utilization overhead for EDP abstractions are 0.0006 and 0.0013, an increase of just 2.48% and 5.23%. The results for the vehicle controller module are similar to those of the battery module. The intuition behind this effect is that

the abstraction process adds an overhead that may not be linear with respect to the demand of the underlying tasks. In other words, a small increase in the demand posed by the underlying task set causes a disproportional abstraction overhead.

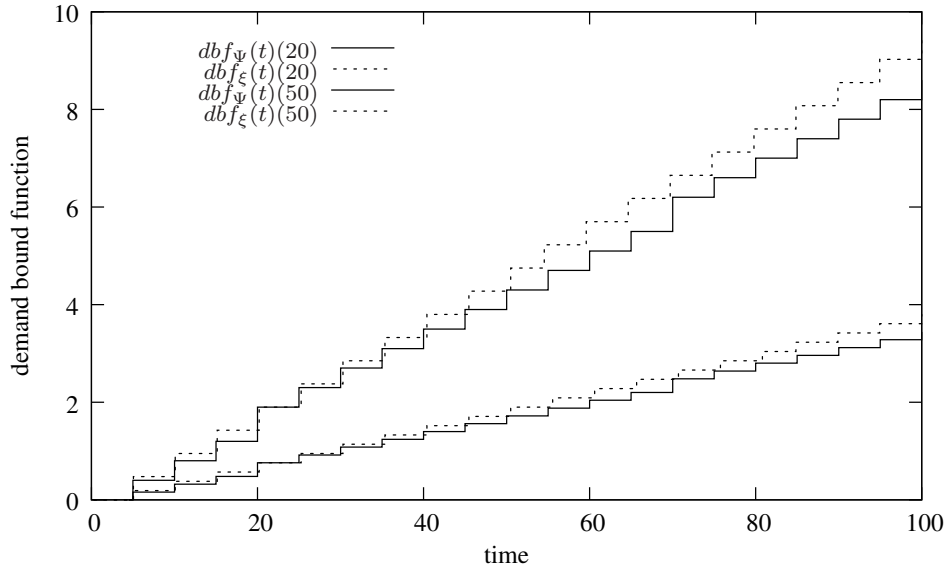


Fig. 18. The EDP abstraction at different bus speeds for the driver module

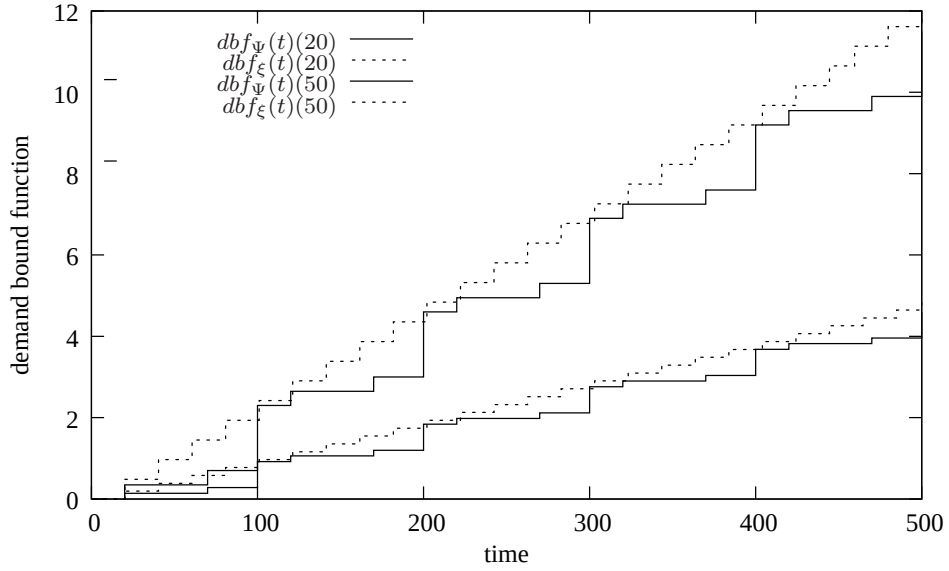


Fig. 19. The EDP abstraction at different bus speeds for the battery module

Abstractions and bus speed: We now investigate the effect of the different abstractions and bus speeds. In our study, we consider two different bus speeds: 20*kbps* and 50*kbps*. Figures 18 and 19

plot the demands for EDP abstractions at these different speeds. As before, dbf_Ψ represents the demand of the module tasks, and dbf_ξ represents the task abstraction corresponding to the EDP resource model. The demand for the conditional task models is provided as a reference. We can make the following observations.

(1) An increase in the bus speed decreases the utilization level and results in less overhead in the abstractions. An increase in the bus speed from 20*kbps* to 50*kbps* results in a 250% increase in speed. Ideally, this would translate into a utilization decrease of $40\%(1/2.5)$ in its value at 20*kbps*. While this was observed for the conditional models in the driver, battery, and vehicle controller modules, the EDP abstractions registered reductions in utilization of 37.63%, 39.41%, and 31.90%, respectively, rather than 40%. These surprisingly low utilizations and the discrepancy between the expected and observed values are most likely due to the fact that as the utilization decreases, the EDP model is better able to abstract the demand of underlying tasks.
(2) The gap between the conditional task abstraction and the EDP task abstraction shrinks as the bus speed increases. This decrease could be due to the increase in the resource supply that decreases the overhead introduced by the EDP task abstraction.
(3) The decrease in the gap between the abstraction demand and the actual demand is consistently observed across modules. For example, the relative utilization overheads between the two abstractions for the driver module at 20*kbps* and 50*kbps* are 10.5% and 3.95%, for the battery module they are 2.48% and 0.98%, and the V/C module they are 42.35% and 13.51%, respectively. These decreases in overhead can be explained by observing that as the bus speed increases, the utilization drops, as does the utilization gap between the conditional and EDP task abstractions.

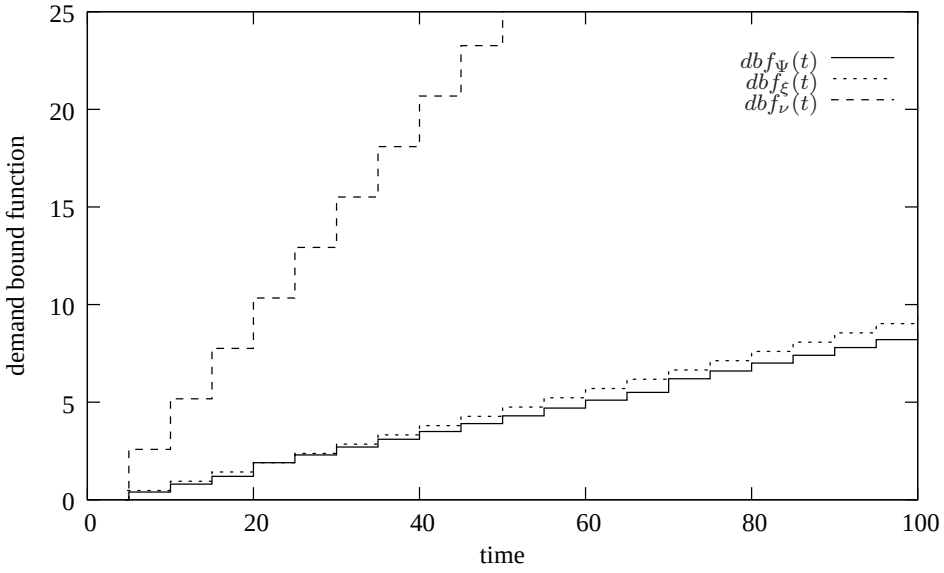


Fig. 20. The abstraction demands for the driver module

The impact of the abstraction technique used: Finally, we compare different abstraction techniques and the overheads they incur. Figures 20, 21, and 22 plot the abstraction demands for the different abstraction techniques. In these plots, dbf_Ψ represents the demand of the module tasks (RTB/RTC), dbf_ξ represent the demand of the task abstraction corresponding to the EDP resource model, and dbf_ν is the task (sporadic) abstraction corresponding to the periodic resource model. As can be seen clearly, the periodic model-based abstraction performed worst out of the three. The

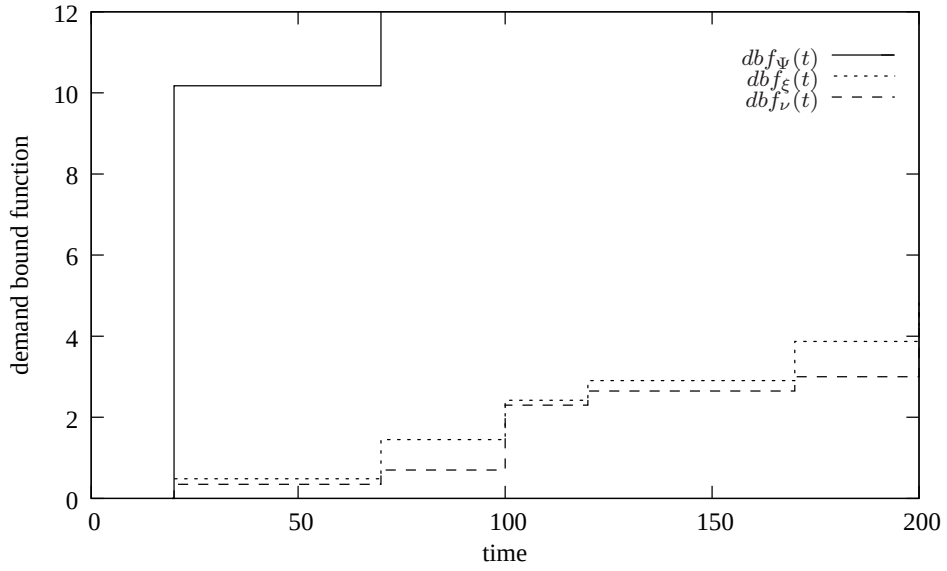


Fig. 21. The abstraction demands for the battery module

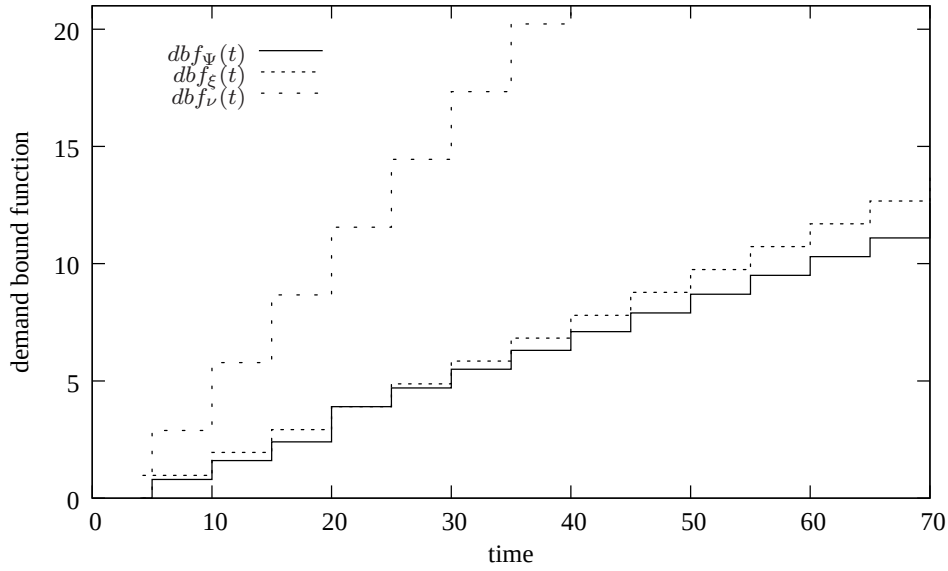


Fig. 22. The abstraction demands for the I/M module

overall demand due to these abstractions is shown in Figure 23. Table III gives the maximum utilization and relative utilization overheads for different abstractions, for each of the modules of the SAE application. The relative utilization overheads are shown as percentages in brackets. As the conditional models do not incur any utilization overhead, the overhead values are omitted in the table. We can make the following observations in this case:

(1) *The conditional task models perform better than other abstractions in terms of utilization.* The tables and figures immediately show this fact. The EDP abstraction of the conditional task

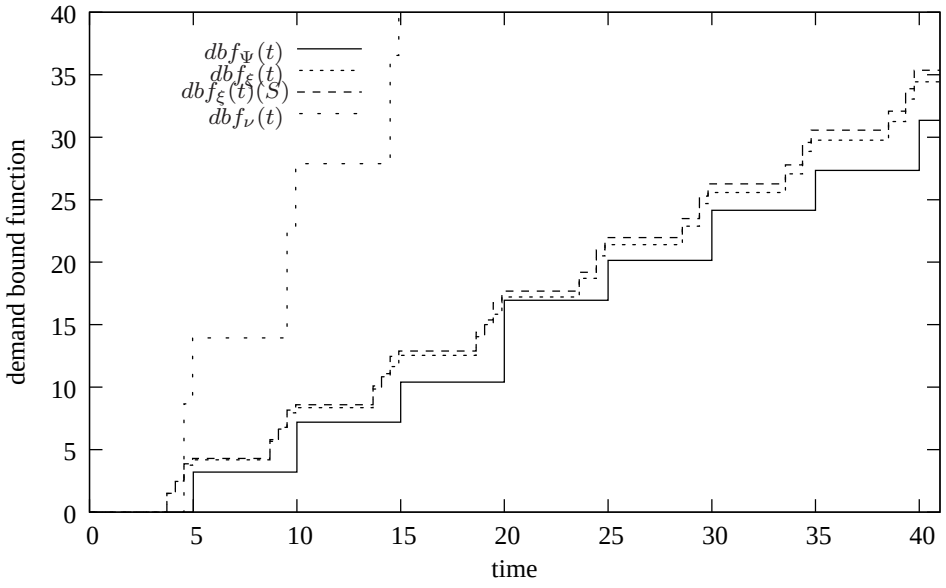


Fig. 23. The abstraction demands for the entire application

Table III. The utilization and overhead for different abstractions

Module	RTB/RTC Abstraction	EDP Abstraction	EDP Abstraction (S)	Periodic Abstraction
Driver	0.0950	0.1050 (10.5%)	0.1268 (33.43%)	0.5620 (491.53%)
Battery	0.0242	0.0248 (2.48%)	0.0255 (5.23%)	0.5178 (2039.72%)
V/C	0.2975	0.4235 (42.35%)	0.4337 (45.78%)	0.6880 (131.25%)
I/M C	0.1950	0.2422 (24.22%)	–	0.6880 (252.81%)
Brakes	0.1650	0.1976 (19.76%)	–	0.6880 (316.96%)
Trans	0.0841	0.0918 (9.18%)	–	0.5845 (594.95%)
Overall	0.8475	0.9134 (7.18%)	0.9380 (10.68%)	3.2972 (289.04%)

models comes close to an overall utilization overhead of 7.18%, and the periodic abstraction performs significantly worse in all cases. In fact, the periodic abstraction with a utilization of 3.2972 is not even schedulable. This lack of schedulability occurs because the periodic model is not able to ensure as tight a supply as the EDP model does. The advantage that the EDP model has with respect to the periodic lies in its explicit deadline. This effect can be clearly seen in Figure 15, which shows both the periodic and EDP abstractions. The EDP model can start earlier and hence, reduce the starvation length and touch the demand curve much later.

(2) *Large periods cause high overprovisioning in the abstraction.* The periodic abstraction suffers greatly for the battery module. This drop in performance occurs because the battery module has a period of 20*ms*, higher than that of other components. Therefore, the periodic resource model ends up providing significantly more resources than are actually needed simply to maintain the minimum utilization.

(3) *The overall utilization overhead for EDP and the periodic resource model is not just the sum of the individual modules.* This property arises because the maximum utilization for each module may occur during different time intervals.

Finally, we discuss the aspects of abstractions other than utilization, namely, the ease of modeling and checking schedulability. Whereas, in terms of utilization, the conditional models perform better, they are harder to model and check for schedulability. Specifically, schedulability checking for conditional models is proportional to utilization. Therefore, in systems with high utilization, the bound on the time interval over which schedulability has to be checked can be very large. In contrast to the conditional models, EDP models appear to incur additional overhead but are comparatively

easier to model and analyze. Periodic models, however, impose too much overhead even if they allow for easy schedulability analysis. Finally, we observed that the utilization of EDP abstractions for sporadic task models is higher than those of conditional models. Therefore, even if conditional task abstractions are not used, the task models may serve as a good first step.

8. CONCLUSIONS

In this paper, we have introduced conditional task models with control variables (RTC) techniques for their compositional analysis. These techniques enable the modeling and analysis of many real-time applications with hierarchical scheduling policies and conditional real-time code. We modeled the SAE class C vehicle communication requirements as a conditional task model and analyzed the message schedulability using various abstraction techniques.

From the results of the study, we conclude that conditional task models can be used to faithfully capture the requirements of a system and yield clear benefits in compositional analysis. Specifically, abstractions based on conditional models incur lower utilization overheads, both individually and for the overall application. Amongst the abstractions compared, conditional task abstractions incurred the least amount of utilization overhead. The EDP model presents a good tradeoff between ease of modeling and the utilization overhead incurred. We conclude by stating some of the limitations of this study and some indications of how they might be overcome.

— We have considered preemptive scheduling here. Solutions involving the CAN bus [Tindell and Burns 1997] support preemptive scheduling across multiple messages. However, many bus technologies support only non-preemptive scheduling, and preemptive scheduling cannot be applied inside network packets. Applying the proposed techniques to a non-preemptive setting usually results in low schedulability, and more work needs to be done in this regard.
— This study was performed using the EDF policy. The results can be easily extended to fixed priority scheduling by modeling the appropriate demand into the tasks. Extending the results to FIFO scheduling may be more difficult, because the queue length for every module would have to be determined precisely.
— In the case study, we did not model overhead, such as header information, which is required in many bus technologies. However, this modeling should be relatively straightforward, and we expect our results to hold also for overheads even though we may see an overall increase in utilization.
— Specific technologies also use optimizations, such as the piggybacking of messages and frame packing. However, such optimizations only affect message utilization and reduce system overhead. We expect the results of our study to hold even if these considerations are explicitly modeled.

REFERENCES

ALMEIDA, L. AND PEDREIRAS, P. 2004. Scheduling Within Temporal Partitions: Response-time Analysis and Server Design. In *EMSOFT '04: Proceedings of the 4th ACM International Conference on Embedded Software*. ACM, New York, NY, USA, 95–103.

ANAND, M. 2008. Conditional models for compostional analysis of real-time embedded systems. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, 19104.

ANAND, M., EASWARAN, A., FISCHMEISTER, S., AND LEE, I. 2008. Compositional feasibility analysis for conditional task models. In *Proceedings of the Eleventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. IEEE Computer Society, Washington, DC, USA.

BARUAH, S. 1998a. Feasibility Analysis of Recurring Branching Tasks. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems (ECRTS)*. 138–145.

BARUAH, S. 1998b. A general model for recurring real-time tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 114–122.

BARUAH, S., HOWELL, R., AND ROSIER, L. 1990. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems 2*, 301–324.

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

Table IV. ETX-I vehicle control system signals

No.	Description	Type	Update Interval	Symbol	From	To
1	Traction Battery Voltage	8	100.0	TBV	Battery	V/C
2	Traction Battery Current	8	100.0	TBI	Battery	V/C
3	Traction Battery Temp, Average	8	1000.0	TBTA	Battery	V/C
4	Auxiliary Battery Voltage	8	100.0	ABV	Battery	V/C
5	Traction Battery Temp, Max.	8	1000.0	TBTM	Battery	V/C
6	Auxiliary Battery Current	8	100.0	ABI	Battery	V/C
7	Accelerator Position	8	5.0	APP	Driver	V/C
8	Brake Pressure, Master Cylinder	8	5.0	BPM	Brakes	V/C
9	Brake Pressure, Line	8	5.0	BPL	Brakes	V/C
10	Transaxle Lubrication Pressure	8	100.0	PLT	Trans	V/C
11	Transaction Clutch Line Pressure	8	5.0	PCT	Trans	V/C
12	Vehicle Speed	8	100.0	WHS	Brakes	V/C
13	Traction Battery Ground Fault	1	1000.0	TGF	Battery	V/C
14	Hi&Lo Contactor Open/Close	4	-		Battery	V/C
15	Key Switch Run	1	-	KSR	Driver	V/C
16	Key Switch Start	1	-	KSW	Driver	V/C
17	Accelerator Switch	2	-	ASW	Driver	V/C
18	Brake Switch	1	-	BSW	Brakes	V/C
19	Emergency Brake	1	-	PBK	Driver	V/C
20	Shift Lever (PRNDL)	3	-		Driver	V/C
21	Motor/Trans Over Temperature	2	1000.0	TOTEMP	Trans	V/C
22	Speed Control	3	-	SPC	Driver	V/C
23	12V Power Ack Vehicle Control	1	-	VCA	Battery	V/C
24	12V Power Ack Inverter	1	-	ICA	Battery	V/C
25	12V Power Ack I/M Contr.	1	-	IMCA	Battery	V/C
26	Brake Mode (Parallel/Split)	1	-	BMD	Driver	V/C
27	SOC Reset	1	-	SOCR	Driver	V/C
28	Interlock	1	-	INT	Battery	V/C
29	High Contactor Control	8	10.0	MHC	V/C	Battery
30	Low Contactor Control	8	10.0	MLC	V/C	Battery
31	Reverse and 2nd Gear Clutches	2	-	PC1/2	V/C	Trans
32	Clutch Pressure Control	8	5.0	PC1/2	V/C	Battery
33	DC/DC Converter	1	1000.0	DDC	V/C	Battery
34	DC/DC Converter Current Control	8	-	DIC	V/C	Battery
35	12V Power Relay	1	-	APC/TPC	V/C	Battery
36	Traction Battery Ground Fault Test	2	1000.0	TGF	V/C	Brakes
37	Brake Solenoid	1	-	BSL	V/C	Brakes
38	Backup Alarm	1	-	BVA	V/C	Brakes
39	Warning Lights	7	-		V/C	Ins.
40	Key Switch	1	-	KSW	V/C	I/M C
41	Main Contactor Close	1	-	MCC	I/M C	V/C
42	Torque Command	8	5.0	TQC	V/C	I/M C
43	Torque Measured	8	5.0	TQM	I/M C	V/C
44	FWD/REV	1	-	FRA	V/C	I/M C
45	FWD/REV Ack.	1	-	FRA	I/M C	V/C
46	Idle	1	-	IDL	V/C	I/M C
47	Inhibit	1	-	SIN	I/M C	V/C
48	Shift in Progress	1	-	SIP	V/C	I/M C
49	Processed Motor Speed	8	5.0	PMS	I/M C	V/C
50	Inverter Temperature Status	2	-	ITS	I/M C	V/C
51	Shutdown	1	-	SDN	I/M C	V/C
52	Status/Malfunction (TBD)	8	-	SML	I/M C	V/C
53	Main Contactor Acknowledge	1	-	MCA	V/C	I/M C

CHAKRABORTY, S., KUNZLI, S., AND THIELE, L. 2003. A general framework for analysing system properties in platform-based embedded system designs. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*. IEEE Computer Society, Washington, DC, USA, 10190.

COMPANY, F. M. 1988. ETX-I final report, vol 1. Tech. rep., Ford.

DAVIS, R. I. AND BURNS, A. 2005. Hierarchical Fixed Priority Pre-Emptive Scheduling. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, Washington, DC, USA, 389–398.

EASWARAN, A., ANAND, M., AND LEE, I. 3-6 Dec. 2007. Compositional Analysis Framework Using EDP Resource Models. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*. 129–138.

FENG, X. A. AND MOK, A. K. 2002. A Model of Hierarchical Real-Time Virtual Resources. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, Los Alamitos, CA, USA.

GHOSAL, A., SANGIOVANNI-VINCENTELLI, A., KIRSCH, C. M., HENZINGER, T. A., AND IERCAN, D. 2006. A Hierarchical Coordination Language for Interacting Real-time Tasks. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded software*. ACM Press, New York, NY, USA, 132–141.

IERCAN, D. 2005. Master's thesis, Politehnica University of Timisoara.

KOPETZ, H. 1994. A Solution to an Automotive Control System Benchmark. Tech. Rep. Research report 4/1994, Institut fur Technische Informatik, TU Wien.

LEHOCZKY, J., SHA, L., AND DING, Y. 1989. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proc. of IEEE Real-Time Systems Symposium*. 166–171.

LIPARI, G. AND BINI, E. 2003. Resource Partitioning Among Real-time Applications. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*. 151–158.

MATIC, S. AND HENZINGER, T. A. 2005. Trading End-to-End Latency for Composability. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, Washington, DC, USA, 99–110.

MEYEROWITZ, T., PINELLO, C., AND SANGIOVANNI-VINCENTELLI, A. 2003. A Tool for Describing and Evaluating Hierarchical Real-time Bus Scheduling Policies. In *Proceedings of the 40th Conference on Design Automation (DAC)*. ACM, New York, NY, USA, 312–317.

NATALE, M. D. AND MESCHI, A. 2001. Scheduling Messages with Earliest Deadline Techniques. *Real-Time Systems Journal 20,* 3, 255–285.

SAE. 1993. Class C Application Requirement Considerations. Tech. Rep. Technical Report J2056/1, SAE.

SAE. 1994. *SAE Handbook 1994 edition*. Society of Automotive Engineers.

SAEWONG, S., RAJKUMAR, R., LEHOCZKY, J., AND KLEIN, M. 2002. Analysis of Hierarchical Fixed-priority Scheduling. In *Proc. of Euromicro Conference on Real-Time Systems*.

SHIN, I. AND LEE, I. 2003. Periodic resource model for compositional real-time guarantees. In *Proc. of IEEE Real-Time Systems Symposium*. 2–13.

SHIN, I. AND LEE, I. 2004. Compositional Real-time Scheduling Framework. In *Proc. of IEEE Real-Time Systems Symposium*.

THIELE, L., CHAKRABORTY, S., AND NAEDELE, M. 2000. Real-time calculus for scheduling hard real-time systems. In *in ISCAS*. 101–104.

THIELE, L., WANDELER, E., AND STOIMENOV, N. 2006. Real-time interfaces for composing real-time systems. In *Proc. of 6th ACM International Conference on Embedded Software*. 34–43.

TINDELL, K. AND BURNS, A. 1997. Guaranteed Message Latencies for Distributed Safety Critical Hard Real-Time Networks. Tech. Rep. YCS 94-229, Dept. Computer Science, Univ. of York, York, UK.

A. ALGORITHMS

A.1. Algorithm to Transform Anisochronous RTB into an Isochronous RTB

In the below 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 updates the sets V, V_F, and E. In Lines 5-21 of Algorithm 2, we insert nodes between v_i and v_0 in Ω such that the duration of the inserted run is $P - \Gamma(r_i)$. This inserted run mimics the transition jitters of the old run r_i (of duration $P - \Gamma(r_i)$), but the demand of each inserted node is the 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. When $\Gamma(r_i) < P - \Gamma(r_i)$, the introduced portion is longer than the old run, and therefore, the remainder demand for $P - 2\Gamma(r_i)$ is inserted in Line 18.

Algorithm 2 RTB-ISO-GEN(Ω, P)

Input: $\Omega = \langle V, v_0, V_F, E, \tau, \rho \rangle$, $\Gamma(r_1) \leq \ldots \leq \Gamma(r_n) \leq P$
Output: Isochronous model Ω

1: Compute $\text{dbf}_\Omega(t), \forall t \leq \max_i\{P - \Gamma(r_i)\}$. Let $d_{max} = 0$.
2: Compute $M = \max_{t < 2\Gamma(r_n)} \frac{\text{dbf}_\Omega(t)}{t}$.
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** $(t \leq P - \Gamma(r_i)) \wedge (u_1 \neq \perp)$ **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 $\Gamma(r_i) \geq P - \Gamma(r_i)$
12: **if** $u_1 \neq v_0$ **then**
13: Create u, $\tau(u) = (M \cdot (P - \Gamma(r_i) - t), P - \Gamma(r_i))$
14: $d_{max} = \max\{d_{max}, \tau(u).e\}$
15: Create $e = \langle u, u_2 \rangle$ s.t. $\rho(e) = P - \Gamma(r_i) - t$
16: $j_\delta = 0$
17: **else**
18: Create u, $\tau(u) = (M \cdot (P - 2\Gamma(r_i)), \Gamma(r_i))$
19: Create $e = \langle u, u_2 \rangle$ s.t. $\rho(e) = 0$.
20: $j_\delta = p - 2\Gamma(r_i)$.
21: **end if**
22: Let $\delta = \max_{t < \Gamma(r_i)}\{\text{rbf}_{r_i}(t) - \text{dbf}_{r_i}(t)\}$
23: Create u_δ, $\tau(u_\delta) = (\delta, P - \Gamma(r_i))$
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 $v_0 = u_{max}$

As we make the RTB model isochronous by introducing new nodes, it is possible that some of the older nodes will 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 RTB, which has a demand equal to the maximum demand over all inserted nodes (except u_δ and the node inserted in Line 18). This is required to handle the case where the critical interval does not end on an inserted node. Whenever $P < 3\Gamma(r_1)$,

all values of dbf and rbf required by the algorithm can be computed in $O(|V|^3)$ time [Baruah 1998a; 1998b], yielding an overall running time of $O(|V|_3 + |V|P)$. In addition, we observe that (1) the number of nodes inserted by the algorithm is $O(|V_F||V|)$ and (2) the reset frame separation property is preserved.

A.2. Algorithm to Transform Anisochronous RTC into an Isochronous RTC Model

For anisochronous RTC models, a procedure similar to RTB-ISO-GEN can be defined to transform them into isochronous RTC models. RTC-ISO-GEN differs 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 the predecessor of the node in the run, (2) existing assignments and enabling conditions for transitions are preserved, and (3) the transition introduced to the new node has no variable assignments and is always enabled. It is required that the Ψ model satisfies the reset frame separation property to ensure that the schedulability analysis remains valid [4]. Additionally, RTC-ISO-GEN only introduces $O|V|$ new nodes in the model because control variables can be used to merge common prefixes of final nodes (e.g., merge nodes introduced in Line 6 of RTB-ISO-GEN).

A.3. Algorithm to Merge Critical Runs of a RTC Model

First, we describe a sub-procedure to merge the critical runs from underlying models, and we will subsequently use this sub-procedure in the final abstraction procedure.

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 the partial runs leading to the node. Then, we extract each node v from this queue and insert it into a run r with an incident transition e from the current node v_c of r. We assume that $\mathsf{l}_{\mathsf{v_c}}$ denotes a unique control variable associated with node $\mathsf{v_c}$. The transition e has the following properties: (1) the enabling condition on e checks whether the variable l_{v_c} is set to v, and (2) the minimum jitter of e is such that the duration up to node v is preserved, i.e., if v belonged to run r_i, then the duration of the partial run leading to v in r_i will be preserved in r. Furthermore, the assignment of 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 24, 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 $r = run(v_a, v_3, 9)$.

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 A.1. *$\Delta(r) = \sum_{i=1}^{m} \Delta(r_i)$ 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. Because 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. However, because v_c precedes v in run r, the duration of the partial run of r leading to v is $t_c + (t_v - t_c) = t_v$. This quantity is equal to the duration of the partial run leading to v in r_i. Because this statement holds for all nodes in r, we get $\Delta(r) = \Delta(r_i)$. □

[4] Without the reset frame separation, Proposition 1 would not hold.

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 $\Gamma(r) = \Gamma(r_i)$.

Create a *min-priority queue* $Q \leftarrow \emptyset$.
for $i = 1$ to m **do**
  for each node v in r_i **do**
    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 $\mathsf{r_i}$ **have higher priority than nodes of** $\mathsf{r_j}$ **for all** $\mathsf{i} < \mathsf{j}$**.**
  end for
end for
Let $v_c =$ 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.
Initialize $r = run(v_c, v_c, 0)$.
while $Q \neq \emptyset$ **do**
  $(t_v, v) =$ EXTRACT-MIN(Q)
  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 $\mathsf{l_{v_c}}$ **denotes a unique control variable associated with node** $\mathsf{v_c}$**.**
  if $e_c \neq \varepsilon$ **then**
    $(t', g', a') = \rho(e_c)$.
    $\rho(e_c) = \langle t', g', \{l_{v_c} = v\}\rangle$
  end if
  $r = r \cdot run(v_c, v, t_v - t_c)$, $e_c = e$.
  $v_c = v$, $t_c = t_v$.
end while

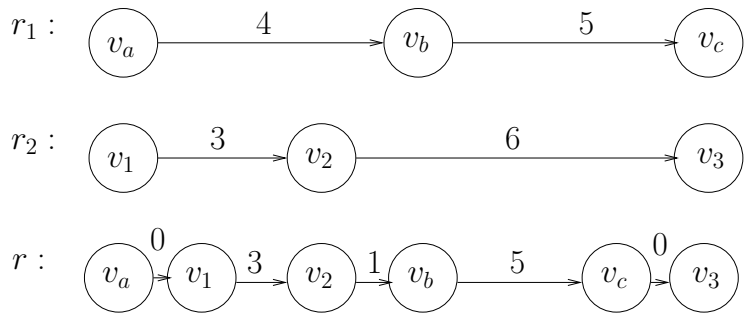


Fig. 24. Example for MERGE