Models for Real-Time Workload: A Survey

Martin Stigge and Wang Yi

Department of Information Technology
Uppsala University, Sweden
Email: {martin.stigge,yi}@it.uu.se

Abstract. This paper provides a survey on task models to characterize real-time workloads at different levels of abstraction for the design and analysis of real-time systems. It covers the classic periodic and sporadic models by Liu and Layland et al., their extensions to describe recurring and branching structures as well as general graph- and automata-based models to allow modeling of complex structures such as mode switches, local loops and also global timing constraints. The focus is on the precise semantics of the various models and on the solutions and complexity results of the respective feasibilty and schedulability analysis problems for preemptable uniprocessors.

1 Introduction

Real-time systems are often implemented by a number of concurrent tasks sharing hardware resources, in particular the execution processors. The designer of such systems need to construct workload models characterizing the resource requirements of the tasks. With a formal description of the workload, a resource scheduler may be designed and analyzed. The fundamental analysis problem to solve in the design process is to check (and thus to guarantee) the schedulability of the workload, i.e., whether the timing constraints on the workload can be met if the scheduler is used. In addition, the workload model may also be used to optimize the resource utilization as well as the average system performance.

In the past decades, workload models have been studied intensively in the theory of real-time scheduling [19] and other contexts, for example performance analysis of networked systems [18]. The research community of real-time systems has proposed a large number of models (often known as task models) allowing for the description and analysis of real-time workloads at different levels of abstraction. One of the classic works is the periodic task model due to Liu and Layland [36, 33, 34], where tasks generate resource requests at strict periodic intervals. The periodic model was extended later to the sporadic model [38, 8, 32] and multiframe models [40, 6] to describe non-regular arrival times of resource requests, and non-uniform resource requirements. In spite of a limited form of variation in release times and worst-case execution times, these repetitive models are highly deterministic. To allow for the description of recurring and non-deterministic behaviours, tree-like recurring models based on directed acyclic graphs are introduced [10, 9] and recently extended to the Digraph model based on arbitrary directed graphs [46, 45] to allow for modeling of complex structures like mode switches and local loops as well as global timing constraints on resource requests. With origin from formal verification of timed systems, the model of task automata [26] was developed in the late 90s.

The essential idea is to use timed automata to describe the release patterns of tasks. Due to the expressiveness of timed automata, it turns out that all the models above can be described using the automata-based model.

In addition to the expressiveness, a major concern in developing these models is the complexity of their analysis. It is not surprising that the more expressive they are, the more difficult to analyze they tend to be. Indeed, the model of task automata is the most expressive model with the highest analysis complexity, which marks the borderline between decidability and undecidability for the schedulability analysis problem of workloads. On top of the operational models summarized above that capture the timed sequences of resource requests representing the system executions, alternative characterizations of workloads using functions on the time interval domain have also been proposed, notably Demand Bound Function (DBF), Request Bound Function (RBF) and Real-Time Calculus (RTC) [51] that can be used to specify the accumulated workload over a sliding time window. They have been further used as a mathematical tool for the analysis of operational workload models.

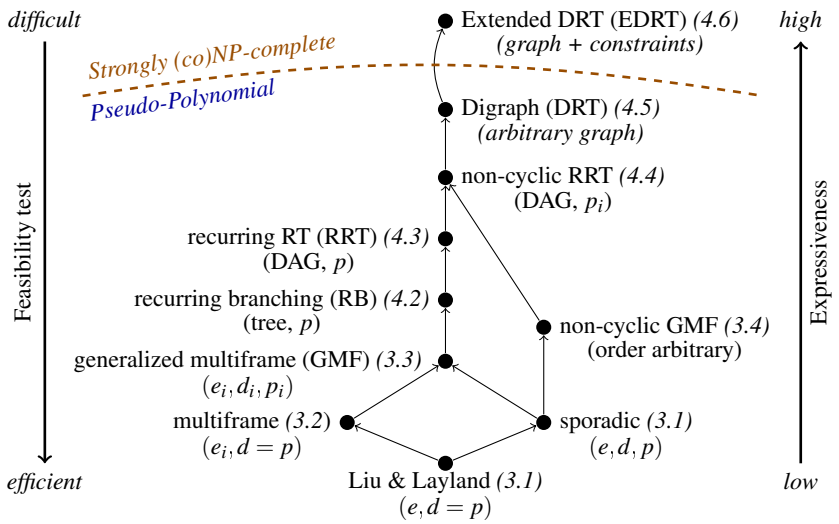


Fig. 1. A hierarchy of task models. Arrows indicate the generalization relationship. The higher in the hierarchy, the higher the expressiveness, but also the more expensive the feasibility test. We denote the corresponding survey section in parentheses.

This paper is intended to serve as a survey on existing workload models for preemptable uniprocessor systems. Figure 1 outlines the relative expressive powers vs. the degree of feasibility analysis difficulty for the models (marked with subsections where they are described). In the following sections we go through this model hierarchy and provide a brief account for each of the well-developed models including the Real-Time Calculus representing models on a higher level of abstraction.

2 Terminology

We first review some terminology used in the rest of this survey. The basic unit describing workload is a *job*, characterized by a *release time*, a *worst-case execution time* (WCET) and a *deadline*. The higher-level structures which generate (potentially infinite) sequences of jobs are *tasks*. The time interval between the release time and the deadline of a job is called its *scheduling window*. The time interval between the release time of a job and the earliest release time of its succeeding job from the same task is its *exclusion window*.

A task system typically contains a set of tasks that at runtime generate sequences of jobs concurrently and a *scheduler* decides at each point in time which of the pending jobs to execute. We distinguish between two important classes of schedulers on preemptable uniprocessor systems:

Static priority schedulers. Tasks are ordered by priority. All jobs generated by a task of higher priority get precedence over jobs generated by tasks of lower priority.

Dynamic priority schedulers. Tasks are not ordered a priori; at runtime, the scheduler may choose freely which of the pending jobs to execute.

Given suitable descriptions of a workload model and a scheduler, one of the important questions is whether all jobs that could ever be generated will meet their deadline constraints, in which case we call the workload *schedulable*. This is determined in a formal *schedulability test* which can be:

Sufficient if it is failed by all non-schedulable task sets,
Necessary, if it is satisfied by all schedulable task sets, or
Precise or exact if it is both sufficient and necessary.

More precisely, if a task set satisfies a sufficient test, it is schedulable and if it fails a necessary test, it is non-schedulable. Workload that is schedulable with *some* effective scheduler is called *feasible*, determined by a feasibility test that can also be either sufficient, necessary or both.

3 Repetitive Models: From Liu and Layland Tasks to GMF

In this section, we show the development of task systems from the periodic task model to different variants of the Multiframe model, including techniques for their analysis.

3.1 Periodic and Sporadic Tasks

The first task model with *periodic* tasks was introduced 1973 by Liu and Layland [36]. Each periodic task $T = (P, E)$ in a task set τ is characterized by a pair of two integers: period P and WCET E. It generates an infinite sequence $\rho = (J_1, J_2, \ldots)$ of jobs $J_i = (r_i, e_i, d_i)$ with release time r_i, execution time e_i and deadline d_i such that $r_{i+1} = d_i = r_i + P$ and $e_i \leqslant E$. This means that jobs are released periodically and have *implicit* deadlines at the release times of the next job.

A relaxation of this model is to allow jobs to be released at later time points, as long as *at least P* time units pass between adjacent job releases of the same task. This is called the *sporadic* task model, introduced by Mok in [38]. Another generalization is to add an explicit deadline D as a third integer to the task definition $T = (P, E, D)$, leading to $d_i = r_i + D$ for all generated jobs. If $D \leqslant P$ for all tasks $T \in \tau$ then we say that τ has *constrained deadlines*, otherwise it has *arbitrary deadlines*.

This model has been the basis for many results throughout the years. Liu and Layland give in [36] a simple feasibility test for implicit deadline tasks: defining the *utilization* $U(\tau)$ of a task set τ as $U(\tau) := \sum_{T_i \in \tau} E_i/P_i$, a task set is uniprocessor feasible if and only if $U(\tau) \leqslant 1$. As in later work, proofs of feasibility are often connected to the *Earliest Deadline First (EDF)* scheduling algorithm, which uses dynamic priorities and has been shown to be optimal for a large class of workload models on uniprocessor platforms. Because of its optimality, EDF schedulability is equivalent to feasibility.

Demand bound function. For the case of explicit deadlines, Baruah et al. [8] introduced a concept that was later called the *demand bound function*: for each interval size t and task T, $dbf_T(t)$ is the maximum accumulated worst-case execution time of jobs generated by T in *any* interval of size t. More specifically, it counts all jobs that have their full scheduling window inside the interval, i.e., release time *and* deadline. The demand bound function $dbf(t)$ of the whole system has the property that a task system is feasible if and only if

$$\forall t. dbf(t) \leqslant t. \tag{1}$$

This condition is a valid test for a very general class of workload models and is of great use in later parts of this survey. It holds for all models generating sequences of independent jobs. A proof can be found in [6].

Focussing on sporadic tasks, Baruah et al. show in [8] that $dbf(t)$ can be computed with

$$dbf(t) = \sum_{T_i \in \tau} E_i \cdot \max\left\{0, \left\lfloor \frac{t - D_i}{P_i} \right\rfloor + 1\right\}. \tag{2}$$

Using this they prove that the feasibility problem is in coNP. Recently it has been shown by Eisenbrand and Rothvoß [23] that the problem is indeed (weakly) coNP-hard for systems with constrained deadlines.

Another contribution of Baruah et al. in [8] was to show that for the case of $U(\tau) < c$ for some constant c, there is a pseudo-polynomial solution of the schedulability problem, by testing Condition (1) for a pseudo-polynomial number of values. The existence of such a constant bound (however close to 1) is a common assumption when approaching this problem since excluding utilizations very close to 1 only rules out very few actual systems.

Static priorities. For static priority schedulers, Liu and Layland show already in [36] that the rate-monotonic priority assignment for implicit deadline tasks is optimal, i.e., tasks with shorter periods have higher priorities. They further give an elegant sufficient schedulability condition by proving that a task set τ with n tasks is schedulable with a static priority scheduler under rate-monotonic priority ordering if

$$U(\tau) \leqslant n \cdot (2^{1/n} - 1). \tag{3}$$

For sporadic task systems with explicit deadlines, the *response time analysis* technique has been developed. It is based on a scenario in which all tasks release jobs at the same time instant with all following jobs being released as early as permitted. This maximizes the response time R_i of the task in question, which is why the scenario is often called the *critical instant*. It is shown by Joseph and Pandya [31] and independently by Audsley et al. [3] that R_i is the smallest positive solution of the recurrence relation

$$R = E_i + \sum_{j<i} \left\lceil \frac{R}{P_j} \right\rceil \cdot E_j, \tag{4}$$

assuming that the tasks are in order of descending priority. This is based on the observation that the interference from a higher priority task T_j to T_i during a time interval of size R can be computed by counting the number of jobs T_j can release as $\lceil R/P_j \rceil$ and multiplying that with their worst-case duration E_j. Together with T_i's own WCET E_i, the response time is derived. Solving Equation (4) leads directly to a pseudo-polynomial schedulability test. Eisenbrand and Rothvoß show in [22] that the problem of computing R_i is indeed NP-hard.

3.2 The Multiframe Model

The first extension of the periodic and sporadic paradigm for jobs of different types, to be generated from the same task was introduced by Mok and Chen in [40]. The motivation is as follows. Assume a workload which is fundamentally periodic but it is known that every k-th job of this task is extra long. As an example, Mok and Chen describe an MPEG video codec that uses different types of video frames. Video frames arrive periodically, but frames of large size and thus large decoding complexity are processed only once in a while. The sporadic task model would need to account for this in the WCET of *all* jobs, which is certainly a significant overapproximation. Systems that are clearly schedulable in practice would fail standard schedulability tests for the sporadic task model. Thus, in scenarios like this where most jobs are close to an average computation time which is significantly exceeded only in well-known periodically recurring situations, a more precise modeling formalism is needed.

To solve this problem, Mok and Chen introduce in [40] the *Multiframe* model. A multiframe task T is described as a pair $(P, \mathbf{E})$ much like the basic sporadic model with implicit deadlines, except that $\mathbf{E} = (E_0, \ldots, E_{k-1})$ is a *vector* of different execution times, describing the WCET of k potentially different *frames*.

Semantics. As before, let $\rho = (J_1, J_2, \ldots)$ be a job sequence with job parameters $J_i = (r_i, e_i, d_i)$ of release time r_i, execution time e_i and deadline d_i. For ρ to be generated by a multiframe task T with k frames, it has to hold that $e_i \leqslant E_{(a+i) \bmod k}$ for some offset a, i.e., the worst-case execution times cycle through the list specified by vector $\mathbf{E}$. The other job parameters r_i and d_i behave as before for sporadic implicit-deadline tasks, i.e., $r_{i+1} \geqslant r_i + P = d_i$. We show an example in Figure 2.

Schedulability Analysis. Mok and Chen provide in [40] a schedulability analysis for static priority scheduling. They provide a generalization of Equation (3) by showing that

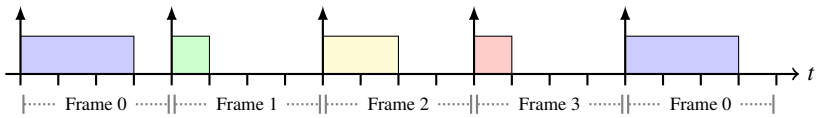


Fig. 2. Example for a Multiframe task $T = (P, \mathbf{E})$ with $P = 4$ and $\mathbf{E} = (3, 1, 2, 1)$. Note that deadlines are implicit.

a task set τ is schedulable with a static priority scheduler under rate-monotonic priority ordering if

$$U(\tau) \leqslant r \cdot n \cdot \left((1 + 1/r)^{1/n} - 1 \right). \tag{5}$$

The value r in this test is the minimal ratio between the largest WCET E_i in a task and its successor $E_{(i+1) \bmod k}$. Note that the classic test for periodic tasks in Equation (3) is a special case of (5) with $r = 1$.

The proof for this condition is done by carefully observing that for a class of Multiframe tasks called *accumulatively monotonic (AM)*, there is a critical instant that can be used to derive the condition (and further even for a precise test in pseudo-polynomial time by simulating the critical instant). In short, AM means that there is a frame in each task such that all sequences starting from this frame always have a cumulative execution demand at least as high as equally long sequences starting from any other frame. After showing (5) for AM tasks the authors prove that each task can be transformed into an AM task which is equivalent in terms of schedulability. The transformation is via a model called General Tasks from [39] which is an extension of Multiframe tasks to an infinite number of frames and therefore of mainly theoretical interest.

Refined sufficient tests have been developed [30, 52, 37] with less pessimism than the test using the utilization bound in (5). They generally also allow certain tasks of higher utilization than those passing the above test to be classified as schedulable. A precise test of exponential complexity is presented in [53] based on response time analysis as a generalization of (4). The authors also include results for models with jitter and blocking.

3.3 Generalized Multiframe Tasks

In the Multiframe model, all frames still have the same period and implicit deadline. Baruah et al. generalize this further in [6] by introducing the *Generalized Multiframe (GMF)* task model. A GMF task $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$ with k frames consists of three vectors:

$\mathbf{P} = (P_0, \ldots, P_{k-1})$ for minimum inter-release separations,
$\mathbf{E} = (E_0, \ldots, E_{k-1})$ for worst-case execution times, and
$\mathbf{D} = (D_0, \ldots, D_{k-1})$ for relative deadlines.

For unambiguous notation we write P_i^T, E_i^T and D_i^T for components of these three vectors in situations where it is not clear from the context which task T they belong to.

Semantics. As a generalization of the Multiframe model, each job $J_i = (r_i, e_i, d_i)$ in a job sequence $\rho = (J_1, J_2, \dots)$ generated by a GMF task T needs to correspond to a frame and the corresponding values in all three vectors. Specifically, we have for some offset a that:

1. $r_{i+1} \geqslant r_i + P_{(a+i) \bmod k}$
2. $e_i \leqslant E_{(a+i) \bmod k}$
3. $d_i = r_i + D_{(a+i) \bmod k}$

An example is shown in Figure 3.

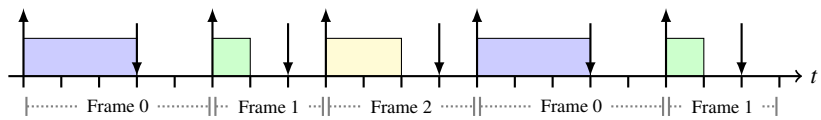


Fig. 3. Example for a GMF task $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$ with $\mathbf{P} = (5,3,4)$, $\mathbf{E} = (3,1,2)$ and $\mathbf{D} = (3,2,3)$.

Feasibility Analysis. Baruah et al. give in [6] a feasibility analysis method based on the demand bound function. The different frames make it difficult to develop a closed-form expression like (2) for sporadic tasks since there is in general no unique critical instant for GMF tasks. Instead, the described method (which we sketch here with slightly adjusted notation and terminology) creates a list of pairs $\langle e, d \rangle$ of workload e and some time interval length d which are called *demand pairs* in later work [46]. Each demand pair $\langle e, d \rangle$ describes that a task T can create e time units of execution time demand during an interval of length d. From this information it can be derived that $dbf_T(d) \geqslant e$ since the demand bound function $dbf_T(d)$ is the *maximal* execution demand possible during *any* interval of that size.

In order to derive all relevant demand pairs for a GMF task, Baruah et al. first introduce a property called *localized Monotonic Absolute Deadlines (l-MAD)*. Intuitively, it means that two jobs from the same task that have been released in some order will also have their (absolute) deadlines in the same order. Formally, this is can be expressed as $D_i \leqslant P_i + D_{(i+1) \bmod k}$, which is more general than the classical notion of constrained deadlines, i.e., $D_i \leqslant P_i$, but still sufficient for the analysis. We assume this property for the rest of this section.

As preparation, the method from [6] creates a sorted list DP of demand pairs $\langle e, d \rangle$ for all i and j each ranging from 0 to $k-1$ with

$$e = \sum_{m=i}^{i+j} E_{m \bmod k}, \quad d = \left(\sum_{m=i}^{i+j-1} P_{m \bmod k} \right) + D_{(i+j) \bmod k}. \tag{6}$$

For a particular pair of i and j, this computes in e the accumulated execution time of a job sequence with jobs corresponding to frames $i, \dots, (i+j) \bmod k$. The value of d is the time from first release to last deadline of such a job sequence. With all these created demand pairs, and using shorthand notation $P_{\text{sum}} := \sum_{i=0}^{k-1} P_i$, $E_{\text{sum}} := \sum_{i=0}^{k-1} E_i$ and

$D_{\min} := \min_{i=0}^{k-1} D_i$, the function $dbf_T(t)$ can be computed with

$$dbf_T(t) = \begin{cases} 0 & \text{if } t < D_{\min}, \\ \max\{e \mid \langle e,d\rangle \in DP \text{ with } d \leqslant t\} & \text{if } t \in [D_{\min}, P_{\text{sum}} + D_{\min}), \\ \left\lfloor \frac{t-D_{\min}}{P_{\text{sum}}} \right\rfloor E_{\text{sum}} + dbf_T\big(D_{\min} + (t - D_{\min}) \bmod P_{\text{sum}}\big) & \text{if } t \geqslant P_{\text{sum}} + D_{\min}. \end{cases} \tag{7}$$

Intuitively, we can sketch all three cases as follows: In the first case, time interval t is shorter than the shortest deadline of any frame, thus not creating any demand. In the second case, time interval t is shorter than $P_{\text{sum}} + D_{\min}$ which implies that at most k jobs can contribute to $dbf_T(t)$. All possible job sequences of up to k jobs are represented in demand pairs in DP, so it suffices to return the maximal demand e recorded in a demand pair $\langle e,d\rangle$ with $d \leqslant t$. In the third case, a job sequence leading to the maximal value $dbf_T(t)$ must include at least one complete cycle of all frames in T. Therefore, it is enough to determine the number of cycles (each contributing E_{sum}) and looking up the remaining interval part using the second case.

Finally, [6] describes how a feasibility test procedure can be implemented by checking Condition (1) for all t at which $dbf(t)$ changes up to a bound

$$D := \frac{U(\tau)}{1 - U(\tau)} \cdot \max_{T \in \tau} \left(P_{\text{sum}}^T - D_{\min}^T\right)$$

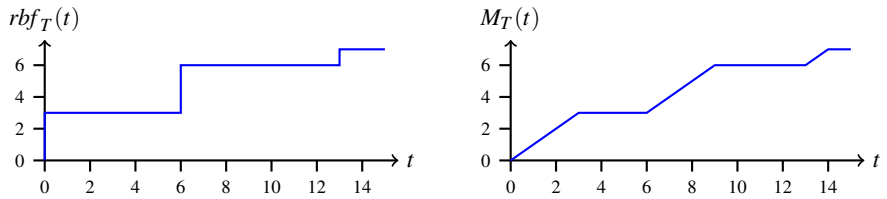
with $U(\tau) := \sum_{T \in \tau} E_{\text{sum}}^T / P_{\text{sum}}^T$ measuring the *utilization* of a GMF task system. If $U(\tau)$ is bounded by a constant $c < 1$ then this results in a feasibility test of pseudo-polynomial complexity. Baruah et al. include also an extension of this method to task systems without the *l-MAD* property, i.e., with arbitrary deadlines. As an alternative test method, they even provide an elegant reduction of GMF feasibility to feasibility of sporadic task sets by using the set DP to construct a dbf-equivalent sporadic task set.

Static priorities. An attempt to solve the schedulability problem for GMF in the case of static priorities was presented by Takada and Sakamura [49]. The idea is to use a function called *maximum interference function (MIF)* $M(t)$. It is based on the *request bound function* $rbf(t)$ which for each interval size t counts the accumulated execution demand of jobs that can be *released* inside any interval of that size. (Notice that in contrast, the demand bound function also requires the job deadline to be inside the interval.) The MIF is a "smoother" version of that, which for each task only accounts for the execution demand that could actually execute inside the interval. We show examples of both functions in Figure 4.

The method uses the MIF as a generalization in the $\sum$-summation term in Equation (4), leading to a generalized recurrence relation for computing the response time:

$$R = E_i + \sum_{j<i} M_j(R) \tag{8}$$

Note that this expresses the response time of a *job* generated by one particular *frame* i of a GMF task with $M_j(t)$ expressing the corresponding maximum interference functions of higher priority tasks. Computation of $M_i(t)$ is essentially the same process as determining the demand bound function $dbf(t)$ from above.



(a) Request bound function (b) Maximum interference function

Fig. 4. Examples for request bound function and maximum interference function of the same task.

It was discovered by Stigge and Yi in [48] that the proposed method does *not* lead to a precise test since the response time computed by solving Equation (8) is overapproximate. The reason is that $M_i(t)$ overapproximates the actual interference caused by higher priority tasks. They give an example in which case the test using (8) determines a task set to be unschedulable while none of the concrete executions would lead to a deadline miss.

Stigge and Yi show in [48] that this is inherent by demonstrating that one single integer-valued function on the time interval domain can not adequately capture the information needed to compute exact response times. Different concrete task sets with different resulting response times need to be abstracted by identical functions, ruling out a precise test. Indeed, they show that the problem of an exact schedulability test for GMF tasks in case of static priority schedulers is strongly coNP-hard implying that there is no adequate replacement for $M_i(t)$. The rather involved proof is mostly focussing on a more expressive task model (DRT, see Section 4.5) but is shown to even hold in the GMF case. Still, the test for GMF presented in [49] is a *sufficient* test of pseudo-polynomial time complexity.

3.4 Non-cyclic GMF

The original motivation for Multiframe and GMF task models was systems consisting of frames with different computational demand and possibly different deadlines and inter-release separation times, arriving in a pre-defined pattern. Consider again the MPEG video codec example where video frames of different complexity arrive, leading to applicability of the Multiframe model. For the presented analysis methods, the assumption of a pre-defined release pattern is fundamental. Consider now a system where the pattern is not known *a priori*, for example if the video codec is more flexible and allows different types of video frames to appear adaptively, depending on the actual video contents. Similar situations arise in cases where the frame order depends on other environmental decisions, e.g. user input or sensor data. These systems can not be modeled with the GMF task model.

Moyo et al. propose in [50] a model called *Non-Cyclic GMF* to capture such behavior adequately. A Non-Cyclic GMF task $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$ is syntactically identical to GMF task from Section 3.3, but with *non-cyclic semantics*. In order to define the semantics formally, let $\phi : \mathbb{N} \to \{0, \ldots, k-1\}$ be a function choosing frame $\phi(i)$ for the i-th job of a job

sequence. Having ϕ, each job $J_i = (r_i, e_i, d_i)$ in a job sequence $\rho = (J_1, J_2, \ldots)$ generated by a non-cyclic GMF task T needs to correspond to frame $\phi(i)$ and the corresponding values in all three vectors:

1. $r_{i+1} \geqslant r_i + P_{\phi(i)}$
2. $e_i \leqslant E_{\phi(i)}$
3. $d_i = r_i + D_{\phi(i)}$

This contains cyclic GMF job sequences as the special case where $\phi(i) = (a+i) \bmod k$ for some offset a. An example of non-cyclic GMF semantics is shown in Figure 5.

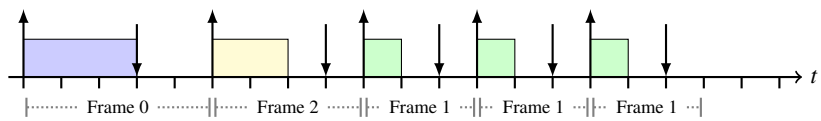


Fig. 5. *Non-cyclic* semantics of the GMF example Figure 3

For analysing non-cyclic GMF models, Moyo et al give in [50] a simple density-based sufficient feasibility test. Defining $D(T) := \max_i C_i^T / D_i^T$ as the *densitiy* of a task T, a task set τ is schedulable if $\sum_{T \in \tau} D(T) \leqslant 1$. This generalizes a similar test for the sporadic task model with explicit deadlines. In addition to this test, [50] also includes an exact feasibility test based on efficient systematic simulation.

A different exact feasibility test is presented in [12] for constrained deadlines using the demand bound function condition (1). A dynamic programming approach is used to compute demand pairs (see Section 3.3) based on the observation that $dbf_T(t)$ can be computed for larger and larger t reusing earlier values. More specifically, a function $A_T(t)$ is defined which denotes for an interval size t the accumulated execution demand of any job sequence where jobs have their full exclusion window inside the interval. It is shown that $A_T(t)$ for $t > 0$ can be computed by assuming that some frame i was the last one in a job sequence contributing a value to $A_T(t)$. In that case, the function value for the remaining job sequence is added to the execution time of that specific frame i. Since frame i is not known a priori, the computation has to take the maximum over all possibilities. Formally,

$$A_T(t) = \max_i \left\{ A_T(t - P_i^T) + E_i^T \mid P_i^T \leqslant t \right\}. \tag{9}$$

Using this, $dbf_T(t)$ can be computed via the same approach by maximising over all possibilities of the last job in a sequence contributing to $dbf_T(t)$. It uses that the execution demand of the remaining job sequence is represented by function $A_T(t)$, leading to

$$dbf_T(t) = \max_i \left\{ A_T(t - D_i^T) + E_i^T \mid D_i^T \leqslant t \right\}. \tag{10}$$

This leads to a pseudo-polynomial time bound for the feasibility test if $U(\tau)$ is bounded by a constant, since $dbf(t) > t$ implies $t < \left(\sum_{T,i} E_i^T\right) / (1 - U(\tau))$ which is pseudo-polynomial in this case.

The same article also proves that evaluating the demand bound function is a (weakly) NP-hard problem. More precisely: Given a non-cyclic GMF task T and two integers t and B it is coNP-hard to determine whether $dbf_T(t) \leqslant B$. The proof is via a rather straightforward reduction from the Integer Knapsack problem. Thus, a polynomial algorithm for computing $dbf(t)$ is unlikely to exist.

Static priorities. A recent result by Berten and Goossens [17] proposes a sufficient schedulability test for static priorities. It is based on the request bound function similar to [49] and its efficient computation. Similar to the approach in [49] the function is inherently overapproximate and the test is of pseudo-polynomial time complexity.

4 Graph-oriented Models

The more expressive workload models become, the more complicated structures are necessary to describe them. In this section we turn to models based on different classes of directed graphs. We start by recasting the definition of GMF in terms of a graph release structure.

4.1 Revisiting GMF

Recall the Generalized Multiframe task model from Section 3.3. A GMF task $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$ consists of three vectors for minimum inter-release separation times, worst-case execution times and relative deadlines of k frames. The same structure can be imagined as a *directed cycle graph*[1] $G = (V, E)$ in which each vertex $v \in V$ represents the release of a job and each edge $(v, v') \in E$ represents the corresponding inter-release separation. A vertex v is associated with a pair $\langle e(v), d(v) \rangle$ for WCET and deadline parameters of the represented jobs. An edge (u, v) is associated with a value $p(u, v)$ for the inter-release separation time.

The cyclic graph structure directly visualizes the cyclic semantics of GMF. In contrast, non-cyclic GMF can be represented by a *complete digraph*. Figure 6 illustrates the different ways of representing a GMF task with both semantics.

4.2 Recurring Branching Tasks

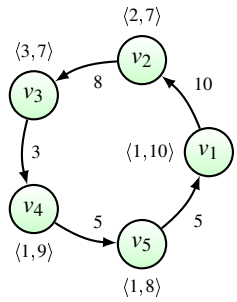
A first generalization to the GMF model was presented in [10]. It is based on the observation that real-time code may include *branches* that influence the pattern in which jobs are released. As the result of some branch, a sequence of jobs may be released which may differ from the sequence released in a different branch. In a schedulability analysis, none of the branches may be universally worse than the others since that may depend on the situation, e.g., which tasks are being scheduled together with the branching one. Thus, all branches need to be modeled explicitly and a proper representation is needed, different from the GMF release structure.

A natural way of representing branching code is a *tree*. Indeed, the model proposed in [10] is a tree representing job releases and their minimum inter-release separation times.

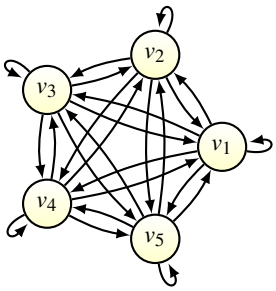
[1] A cycle graph is a graph consisting of one single cycle, i.e., one closed chain.

$T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$
$\mathbf{P} = (10, 8, 3, 5, 5)$
$\mathbf{E} = (1, 2, 3, 1, 1)$
$\mathbf{D} = (10, 7, 7, 9, 8)$

(a) T as vectors



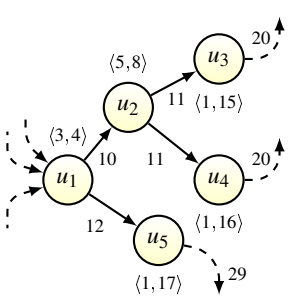
(b) T as cycle graph



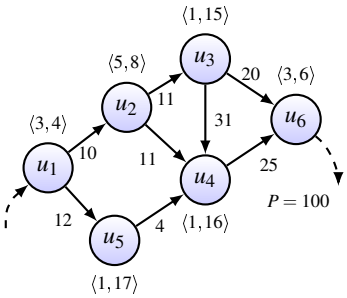
(c) T as complete graph

Fig. 6. Different ways of representing a GMF task T. The vector-representation in 6(a) from Section 3.3 does by itself not imply cyclic or non-cyclic semantics. This is more clear with graphs in 6(b) and 6(c). Note that we omit vertex and edge labels in 6(c) for clarity.

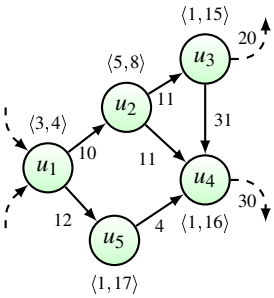
We show an example in Figure 7(a). Formally, a *Recurring Branching (RB)* task T is a directed tree $G(T) = (V, E)$ in which, as in Section 4.1, each vertex $v \in V$ represents a type of job to be released and each edge $(u, v) \in E$ the minimum inter-release separation times. They have labels $\langle e(v), d(v) \rangle$ and $p(u, v)$ as before. In addition to the tree, each leaf u has a separation time $p(u, v_{root})$ to the root vertex v_{root} in order to model that the behavior *recurs* after each traversal of the tree.



(a) RB task



(b) RRT task



(c) Non-cyclic RRT task

Fig. 7. Examples for RB, RRT and non-cyclic RRT task models

In order to simplify the feasibility analysis, the model is syntactically restricted in the following way. For each path $\pi = (v_0, \ldots, v_l)$ of length l from the root $v_0 = v_{root}$ to a leaf v_l, its duration when going back to the root must be the same, i.e., the value

$P := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + p(v_l, v_{root})$ must be independent of π. We call P the *period* of T. Note that this is a generalization of GMF since GMF can be expressed as a linear tree.

Semantics. A job sequence $\rho = (J_1, J_2, \dots)$ is generated by an RB task T if it corresponds to a path π through $G(T)$ in the following way. Path π starts at some vertex v_1 in $G(T)$, follows the edges to a leaf, then starts again at the root vertex, traverses $G(T)$ again in a possibly different way, etc. (Very short π may of course never reach a leaf.) The correspondence between ρ and π means that for all $J_i = (r_i, e_i, d_i)$, we have:

1. $r_{i+1} \geqslant r_i + p(v_i, v_{i+1})$,
2. $e_i \leqslant e(v_i)$,
3. $d_i = r_i + d(v_i)$.

Feasibility Analysis. The analysis presented in [10] is based on the concept of *demand pairs* as described before. We sketch the method from [10] in a slightly adjusted manner. First, a set DP_0 is created consisting of all demand pairs corresponding to paths not containing both a leaf and a following root vertex. This is straight forward since for each pair of vertices (u, v) in $G(T)$ connected by a directed path π, this connecting path is unique. Thus, a demand pair $\langle e, d \rangle$ can be created by enumerating all vertex pairs (u, v) and computing for their connecting path $\pi = (v_0, \dots, v_l)$ the values

$$e := \sum_{i=0}^{l} e(v_i), \quad d := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + d(v_l). \tag{11}$$

Second, all paths π which *do* contain both a leaf and a following root vertex can be cut into three subpaths π_{head}, π_{middle} and π_{tail}:

$$\pi = (\underbrace{v, \dots, v_l}_{\pi_{head}}, \underbrace{v_{root}, v', \dots, v'_l, v_{root}, v'', \dots, v''_l}_{\pi_{middle}}, \underbrace{v_{root}, v''', \dots, v'''')}_{\pi_{tail}}$$

We use v_l, v'_l, etc. for arbitrary leaf nodes. The first part π_{head} is the prefix of π up to and including the first leaf in π. The second part π_{middle} is the middle part starting with v_{root} and ending in the last leaf which π visits. Note that π_{middle} may traverse the tree several times. The third part π_{tail} starts with the last occurrence of v_{root} in π. For each of the three parts, a data structure is created so that demand pairs for a full path π can be assembled easily.

For representing π_{head}, a set UP_{leaf} is created. For all paths $\pi_{head} = (v_0, \dots, v_l)$ that end in a leaf it contains a pair $\langle e, p \rangle$ with

$$e := \sum_{i=0}^{l} e(v_i), \quad p := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + p(v_l, v_{root}). \tag{12}$$

For representing π_{middle}, the maximal accumulated execution demand $e_{\max}$ of any path completely traversing the tree is computed. Note that all paths from the root to a leaf have the same sum of inter-release separation times and this sum is the period P of T. Finally, for representing π_{tail}, a set DP_{root} is computed as a subset of DP_0 only considering paths starting at v_{root}.

Using these data structures, $dbf_T(t)$ can be computed easily. If $t \leqslant P$, then a job sequence contributing to $dbf_T(t)$ either corresponds to a demand pair in DP_0 (not passing v_{root}) or is represented by items from UP_{leaf} and DP_{root} (since it is passing v_{root} exactly once):

$$F_1(t) = \max\{e \mid \langle e, d\rangle \in DP_0 \text{ with } d \leqslant t\} \tag{13}$$

$$F_2(t) = \max\{e_1 + e_2 \mid \langle e_1, p\rangle \in UP_{leaf} \wedge \langle e_2, d\rangle \in DP_{root} \wedge p + d \leqslant t\} \tag{14}$$

$$dbf_T(t) = \max\{F_1(t),\ F_2(t)\} \quad \text{if } t \leqslant P \tag{15}$$

In case $t > P$, such a job sequence *must* pass through v_{root} and traverses the tree completely for either $\lfloor t/P \rfloor$ or $\lfloor t/P \rfloor - 1$ times. For the parts that can be represented by π_{head} and π_{tail} of the corresponding path π, we can use $dbf_T(t)$ from Equation (15), since π_{head} concatenated with π_{tail} correspond to a job sequence without a complete tree traversal. Putting it together for $t > P$:

$$F_3(t) = \left\lfloor \frac{t}{P} \right\rfloor \cdot e_{\max} + dbf_T(t \bmod P) \tag{16}$$

$$F_4(t) = \left\lfloor \frac{t-P}{P} \right\rfloor \cdot e_{\max} + dbf_T((t-P) \bmod P) \quad \text{if } t \geqslant P, 0 \text{ otherwise} \tag{17}$$

$$dbf_T(t) = \max\{F_3(t),\ F_4(t)\} \quad \text{if } t > P \tag{18}$$

Finally, in order to do the feasibility test, i.e., verify Condition (1), the demand bound function $dbf(t) = \sum_T dbf_T(t)$ is computed for all t up to a bound D derived in a similar way as for GMF in Section 3.3.

4.3 Recurring Real-Time Tasks (RRT) – DAG structures

In typical branching code, the control flow is joined again after the branches are completed. Thus, no matter which branch is taken, the part after the join is common to both choices. In the light of a tree release structure as in the RB task model above, this means that many vertices in the tree may actually represent the same types of jobs to be released, or even whole subtrees are equal. In order to make use of these redundancies, Baruah proposes in [9] to use a *directed acyclic graph (DAG)* instead of a tree. The impact is mostly efficiency: each DAG can be unrolled into a tree, but that comes at the cost of potentially exponential growth of the graph.

A *Recurring Real-Time Task (RRT) T* is a directed acyclic graph $G(T)$. The definition is very similar to RB tasks in the previous section, we only point out the differences. It is assumed that $G(T)$ contains one unique *source* vertex (corresponding to v_{root} in an RB task) and further one unique *sink* vertex (corresponding to leafs in a tree). An RRT task has an explicitly defined period parameter P that constrains the minimum time between two releases of jobs represented by the source vertex. An RRT behaves just like an RB task by following paths through the DAG. We skip the details and give an example of an RRT task in Figure 7(b).

Feasibility Analysis. Because of its close relation to RB tasks, the feasibility analysis method presented in [9] is very similar to the method presented above for RB tasks and

we skip the details. However, the adapted method has exponential complexity since it enumerates paths explicitly.

Chakraborty et al. present a more efficient method in [20] based on a dynamic programming approach, leading back to pseudo-polynomial complexity. Instead of enumerating all pairs of vertices (u,v) in the DAG, the graph is traversed in a breadth-first manner. The critical observation is that all demand pairs representing a path ending in any particular vertex v can be computed from those for paths ending in all parent vertices. It is *not* necessary to have precise information about which the actual paths are that the demand pairs represent. Even though Chakraborty et al. consider a limited variant of the model in which paths traverse the DAG only once, the ideas can be applied in general to the full RRT model.

The feasibility problem is further shown to be NP-hard in [20] via a reduction from the Knapsack problem and the authors give a fully polynomial time approximation scheme. For the special case where all vertices have equal WCET annotations, they show a polynomial time solution, similar to the dynamic programming technique above.

Static Priorities. A sufficient test for schedulability of an RRT task set with static priorities is presented in [11]. It is shown that, up to a polynomial factor, the *priority assignment* problem in which a priority order has to be found is equivalent to the *priority testing* problem where a task set with a given priority order is to be tested for schedulability. At the core of both is the test whether a given task $T \in \tau$ will meet its deadlines if it has the lowest priority of all tasks in τ (whose relative priority order does not matter). In that case T is called *lowest-priority feasible*.

The proposed solution gives a condition involving both the demand bound function $dbf_T(t)$ and the request bound function $rbf_T(t)$. It is shown that a task T is lowest-priority feasible if

$$\forall t.\ \ \exists t' \leqslant t.\ \ t' \geqslant dbf_T(t) + \sum_{T' \in \tau \setminus \{T\}} rbf_{T'}(t'). \tag{19}$$

It is shown that $rbf_T(t)$ can be computed with just a minor modification to the computation procedure of $dbf_T(t)$ and that Condition (19) only needs to be checked for a bounded *testing set* of t, similar to the bound D introduced in checking Condition (1) in feasibility tests. For each t, checking the existence of a $t' \leqslant t$ is essentially identical to an iterative procedure of solving the recurrence relation in Equation (8) of which (19) is a generalization.

A tighter and more efficient test is shown in [20] based on a smoother variant of the request bound function, denoted $rbf'_T(t)$. Using this, a task T is lowest-priority feasible if

$$\forall v \in G(T).\ \ \exists t' \leqslant d(v).\ \ t' \geqslant e(v) + \sum_{T' \in \tau \setminus \{T\}} rbf'_{T'}(t'). \tag{20}$$

This test is a more direct and tighter generalization of the sufficient test (8) for GMF tasks.

4.4 Non-cyclic RRT

A further generalization of RRT is *non-cyclic RRT*[2] [13] where the assumption of one single sink vertex is removed. Specifically, a non-cyclic RRT task T is a DAG $G(T)$ with vertex and edge labels as before that has a unique source vertex v_{source}. Additionally, for every sink vertex v, there is a value $p(v, v_{source})$ as before. We give an example in Figure 7(c). Note that a non-cyclic RRT task does *not* have a general period parameter, i.e., paths through $G(T)$ visiting v_{source} repeatedly may do so in differing time intervals when doing so through different sinks.

Feasibility Analysis. The analysis technique presented in [13] is similar to the ones of RB and RRT. The author uses the dynamic programming technique from [20] to compute demand pairs inside the DAG in order to keep pseudo-polynomial complexity and assumes a partition of paths π into π_{head}, π_{middle} and π_{tail} as before. The difference here is that paths traversing $G(T)$ completely from v_{source} to a sink may have different lengths, i.e., π_{middle} is not necessarily a multiple of some period P. Thus, the expressions for partial $dbf_T(t)$ computation in (16) and (17) can't just assume a fixed length P and a fixed computation time $e_{\max}$. The idea to solve this is to first use the technique from [20] to compute demand pairs for full DAG traversals. These can then be interpreted as *frames* with a length and an execution time requirement, which can be concatenated to achieve a certain interval length, like a very big non-cyclic GMF task. Similar to (9) for solving non-cyclic GMF feasibility, all possible paths going from source to a sink can be represented in a function $A_T(t)$ that expresses for each t the amount of execution demand these special paths may create during intervals of length t. Similar to (10), this function is integrated into (16) and (17), resulting in an efficient procedure.

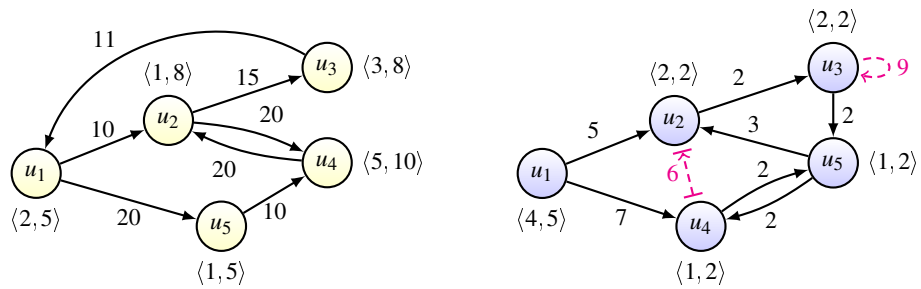
The procedure is generalized in the following section which generalizes and unifies all feasibility tests presented so far.

4.5 Digraph Real-Time Tasks

Stigge et al. observe in [46] that the non-cyclic RRT model can be generalized to *any* directed graph. They introduce the *Digraph Real-Time (DRT)* task model and describe a feasibility test of pseudo-polynomial complexity for task systems with utilization bounded by a constant. A DRT task T is described by a directed graph $G(T)$ with edge and vertex labels as before. There are no further restrictions, any directed graph can be used to describe a task. Using any graph allows to model local loops which was not possible in any model presented above. Even in the non-cyclic RRT model, all cycles in that model have to pass through the source vertex. An example of a DRT task is shown in Figure 8(a).

Semantics. The behavior of a DRT task T is similar to earlier models. A job sequence $\rho = (J_1, J_2, \ldots)$ is generated by T if there is a path $\pi = (v_1, v_2, \ldots)$ through $G(T)$ such that for each job $J_i = (r_i, e_i, d_i)$ it holds that

[2] The name "non-cyclic RRT" can be a bit misleading. The behavior of a non-cyclic RRT task *is* cyclic, in the sense that the source vertex is visited repeatedly. However, in comparison to the RRT model, the behavior is *non-periodic*, in the sense that revisits of the source vertex may happen in different time intervals.



(a) DRT task (b) EDRT task

Fig. 8. Examples for DRT and EDRT task models. The EDRT task in Figure 8(b) contains two additional constraints $(u_4, u_2, 6)$ and $(u_3, u_3, 9)$, denoted with dashed arrows. Note that these dashed arrows do *not* represent edges that can be taken. They only denote additional timing constraints.

1. $r_{i+1} \geqslant r_i + p(v_i, v_{i+1})$,
2. $e_i \leqslant e(v_i)$, and
3. $d_i = r_i + d(v_i)$.

Note that this implies sporadic job releases as before. However, worst-case sequences usually release jobs as soon as permitted.

Feasibility Analysis. Stigge et al. present in [46] a feasibility analysis method which is again based on demand pairs. However, there is no single vertex through which all paths pass and neither does the graph $G(T)$ represent a partial order as DAGs do. Thus, earlier dynamic programming approaches can not be applied. It is not possible either to simply enumerate all paths through the graph since that would lead to an exponential procedure. Instead, the authors propose a *path abstraction* technique which is essentially also based on dynamic programming, generalizing earlier approaches.

For the case of constrained deadlines, each path $\pi = (v_0, \ldots, v_l)$ through $G(T) = (V, E)$ is abstracted using a *demand triple* $\langle e(\pi), d(\pi), v_l \rangle$ with

$$e(\pi) := \sum_{i=0}^{l} e(v_i), \quad d(\pi) := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + d(v_l). \tag{21}$$

It contains as first two components a demand pair and the third component is the last vertex in the path. This abstraction allows to create all demand pairs for $G(T)$ iteratively. The procedure starts with all demand triples that represent paths of length 0, i.e., with the set

$$\{\langle e(v), d(v), v \rangle \mid v \in V\}. \tag{22}$$

In each step, a triple $\langle e, d, v \rangle$ is picked from the set and extended to create new triples $\langle e', d', v' \rangle$ via

$$e' := e + e(v'), \quad d' := d - d(v) + p(v, v') + d(v'), \quad (v, v') \in E. \tag{23}$$

This is done for all edges $(v, v') \in E$. The procedure abstracts from concrete paths since the creation of each new triple $\langle e', d', v' \rangle$ does *not* need the information of the full path represented by $\langle e, d, v \rangle$. Instead, the last vertex v of any such path suffices. The authors show that this procedure is efficient since it only needs to be executed *once* up to a bound D as before and the number of demand triples is bounded.

Further contributions of [46] include an extension of the method to arbitrary deadlines and a few optimisation suggestions for implementations. One of them is considering *critical* demand triples $\langle e, d, v \rangle$ for which no other demand triple $\langle e', d', v' \rangle$ exists with

1. $e' \geqslant e$,
2. $d' \leqslant d$, and
3. $v' = v$.

It is shown that only critical demand triples need to be stored during the procedure. All other, non-critical, demand triples can be discarded since they and all their future extensions will be *dominated* by others when considering contributions to the demand bound function $dbf_T(t)$. This optimization reduces the cubic time complexity of the algorithm to about quadradic in the task parameters.

The presented method is applicable to *all* models in the previous sections since DRT generalizes all of them. In a few special cases, custom optimizations may speed up the process.

Static Priorities. Even though sufficient tests involving $rbf(t)$ and $dbf(t)$ similar to Conditions (19) and (20) can be applied to DRT as well, no exact schedulability test for DRT with static priorities is currently known. It was shown in [48] that the problem is strongly NP-hard via a reduction from the 3-PARTITION problem. This implies that an exact test with pseudo-polynomial complexity is impossible, assuming $\mathsf{P} \neq \mathsf{NP}$. Further, even fully polynomial-time approximation schemes can not exist either. The result holds for all models at least as expressive as GMF.

4.6 Global Timing Constraints

In an effort to investigate the border of how far graph-based workload models can be generalized before the feasibility problem becomes infeasible, Stigge et al. propose in [45] a task model called *Extended DRT (EDRT)*. In addition to a graph $G(T)$ as in the DRT model, a task T also includes a set $C(T) = \{(from_1, to_1, \gamma_1), \ldots, (from_k, to_k, \gamma_k)\}$ of *global inter-release separation constraints*. Each constraint $(from_i, to_i, \gamma_i)$ expresses that between the visits of vertices $from_i$ and to_i, at least γ_i time units must pass. An example is shown in Figure 8(b).

Feasibility Analysis. The feasibility analysis problem for EDRT indeed marks the tractability borderline. In case the number of constraints in $C(T)$ is *bounded* by a constant k which is a restriction called k-EDRT, [45] presents a pseudo-polynomial feasibility test. If the number is not bounded, they show that feasibility analysis becomes strongly NP-hard by a reduction from the Hamiltonian Path problem, ruling out pseudo-polynomial methods.

For the bounded case, we illustrate why the iterative procedure described in Section 4.5 for DRT can not be applied directly and then sketch a solution approach. The

demand triple method can not be applied without change since the abstraction loses too much information from concrete paths. In the presence of global constraints, the procedure needs to keep track of which constraints are *active*. Consider path $\pi = (u_4, u_5)$ from the example in Figure 8(b), which would be abstracted with demand triple $\langle 2, 4, u_5 \rangle$. An extension with u_2 leading to demand triple $\langle 4, 7, u_2 \rangle$ is *not* correct, since the path $\pi' = (u_4, u_5, u_2)$ includes a global constraint, separating releases of the jobs associated with u_4 and u_2 by 6 time units. A correct abstraction of π' would therefore be $\langle 4, 8, u_2 \rangle$, but it is impossible to construct that from $\langle 2, 4, u_5 \rangle$ which lost information about the active constraint.

A solution to this problem is to integrate information about active constraints into the analysis. For each constraint $(from_i, to_i, \gamma_i)$, the demand triple is extended by a *countdown* that represents how much time must pass until to_i may be visited again. This slows down the analysis, but since the number of constraints is bounded by a constant, the slowdown is only a polynomial factor. Stigge et al. choose in [45] to not directly integrate the countdown information into the iterative procedure but to transform each EDRT task T into a DRT task T' where the countdown information is integrated into the vertices. The transformation leads to an equivalent iterative procedure and has the advantage that an already existing graph exploration engine for feasibility tests of DRT task sets can be reused without any change. The transformation is illustrated in Figure 9.

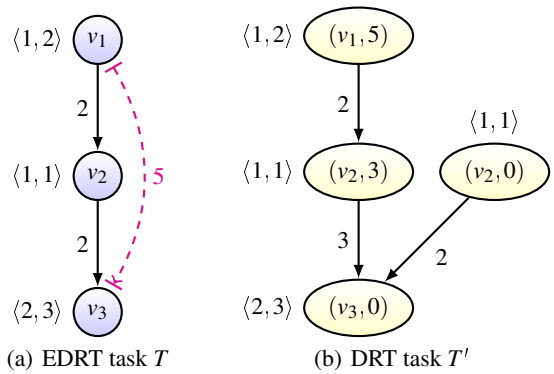


Fig. 9. A basic example of a 1-EDRT task T being transformed into an equivalent DRT task T'.

5 Beyond DRT

We now turn to models that either extend the DRT model in other directions or are outside the hierarchy in Figure 1 since they operate on a different abstraction level.

5.1 Task Automata

A very expressive workload model called task automata is presented by Fersman et al. in [28]. It is based on Timed Automata that have been studied thoroughly in the context of formal verification of timed systems [1, 16]. Timed automata are finite automata extended with real-valued clocks to specify timing constraints as enabling conditions, i.e., guards on transitions. The essential idea of task automata is to use the timed language of an automaton to describe task release patterns.

In DRT terms, a task automaton (TA) T is a graph $G(T)$ with vertex labels as in the DRT model, but labels on edges are more expressive. An edge (u,v) is labeled with a *guard* $g(u,v)$ which is a boolean combination of clock comparisons of the form $x \bowtie C$ where C is a natural number and $\bowtie \in \{\leqslant, <, \geqslant, >\}$. Further, an edge may be labeled with a *clock reset* $r(u,v)$ which is a set of clocks to be reset to 0 when this edge is taken. Since the value of clocks is an increasing real value which represents that time passes, guards and resets can be used to constrain timing behavior on generated job sequences. We give an example of a task automaton in Figure 10.

A task automaton has an initial vertex. In addition to resets and guards, task automata have a synchronization mechanism. This allows them to synchronize on edges either with each other or with the scheduler on a job finishing time.

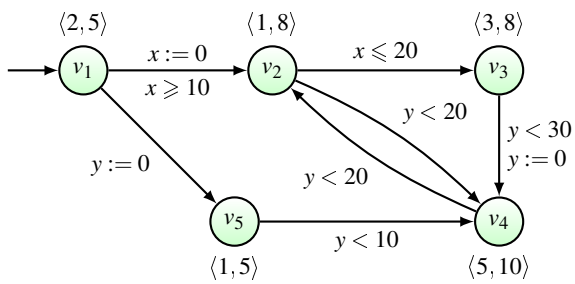


Fig. 10. Example for a task automaton.

Note that DRT tasks are special cases of task automata where only one clock is used. Each edge (u,v) in a DRT task with label $p(u,v) = C$ can be expressed with an edge in a task automaton with guard $x \geqslant C$ and reset $x := 0$.

The authors of [26] show that the schedulabiliy problem for a large class of systems modeled as task automata can be solved via a reduction to a model checking problem for ordinary Timed Automata. A tool for schedulability analysis of task automata is presented in [2, 27]. In fact, the feasibility problem is decidable for systems where at most two of the following conditions hold:

Preemption. The scheduling policy is preemptive.
Variable Execution Time. Jobs may an interval of possible execution times.
Task Feedback. The finishing time of a job may influence new job releases.

However, the feasibility problem is undecidable if all three conditions are true [26]. Task automata therefore mark a borderline between decidable and undecidable problems for workload models.

5.2 Real-Time Calculus

A more abstract formalism to model real-time workload is the *Real-Time Calculus (RTC)* introduced in [51]. Its abstractions are based on the notions of

Event streams, representing the workload,
Computing capacity, representing the computing resource for the workload, and
Processing units, modeling the process of the workload being executed, using up computing capacity.

Formally, a request function $R(t)$ models the accumulated amount of workload up to a time point t, and similarly, a capacity function $C(t)$ models the amount of computation resource available until time t. Both functions are the inputs to a processing unit, which outputs a new event stream modeled by a function $R'(t)$, and leaves remaining computing capacity modeled by $C'(t)$. The four functions are related by

$$R'(t) = \min_{0 \leqslant u \leqslant t} \{R(u) + C(t) - C(u)\}, \tag{24}$$

$$C'(t) = C(t) - R(t). \tag{25}$$

Modeling static priority scheduling is rather straight forward in this model by creating a number of processing units which all have their own input of request functions (representing different tasks) but all being chained through the corresponding capacity functions, i.e., $C'(t)$ of one unit being $C(t)$ of the next one. Other schedulers can also be modeled, with more involved constructions.

A concrete system may create many different request functions $R(t)$. In order to represent these, Real-Time Calculus considers upper and lower bounds in the time interval domain. Specifically, for abstracting the request function $R(t)$, two *arrival curves* $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ are introduced, so that for all $t \geqslant 0$ and $\Delta \geqslant 0$

$$\alpha^l(\Delta) \leqslant R(t+\Delta) - R(t) \leqslant \alpha^u(\Delta). \tag{26}$$

Clearly, a pair (α^u, α^l) abstracts an infinite set of event streams. Analogously, a pair of *service curves* (β^u, β^l) is defined in the time domain to abstract computing capacity.

All analysis can be executed on the level of arrival and service curves. For example, the remaining service curve can be computed via

$$\beta^{l\prime}(\Delta) = \max_{0 \leqslant u \leqslant \Delta} \left\{ \beta^l(u) - \alpha^u(u) \right\}. \tag{27}$$

Response-time analysis can be performed by deriving the response time from the horizontal distance between α^u and β^l.

For many classes of systems, arrival curves can be specified directly with closed form expressions. This includes the sporadic task model with an expression similar to (2), but is also possible for related models like periodic events with jitter [42].

It is worth noting that arrival curves and the request bound function introduced earlier are equivalent concepts. However, RTC models systems are directly using arrival curves, i.e., they are the low-level construct used for system representation. In contrast, the request bound function (and similarly, the demand bound function) is a tool for schedulability analysis that abstracts from a more concrete system description. It is a potentially overapproximate abstraction and the process of deriving request and demand bound functions for task models presented in Sections 3 and 4 is more or less computationally demanding. Thus, the advantage of RTC is that arrival curves can be assumed as given input to the analysis procedures, at the cost of introducing imprecision.

6 Conclusions and Outlook

This survey is by no means complete. In preparing this paper, we have intended to cover only works on independent tasks for preemptive uniprocessors. Other sources for a survey on related topics can be found in [4] on the broad area of static-priority scheduling and analysis, [7] on scheduling and analysis of repetitive models for preemptable uniprocessors, [21] on real-time scheduling for multiprocessors and [44] on the historic development of real-time systems research in general. To complement the survey provided by [7], we have added recent work on models with richer structures, that are based on graphs and automata as well as the Real-Time Calculus based on functions. Apart from their importance in theoretical studies, we believe that these expressive models may find their applications in for example model- and component-based design for timed systems.

The following is a list of areas that we consider important, but did not include them in this paper due to time and page limits.

Resource Sharing. Real-time systems even on uniprocessor platforms contain not only the execution processor, but often many resources shared by concurrently running jobs using a locking mechanism during their execution. Locking can introduce unwanted effects like deadlocks and priority inversion. For periodic and sporadic task systems, the Priority Ceiling (and Inheritance) Protocols [43] and Stack Resource Policy [5] are established solutions with efficient schedulability tests. While these protocols can be used for more expressive models like DRT, the analysis methods do not necessarily apply. Some initial work exists on extending the classical results to graph-based models, e.g., [29] which develops a new protocol to deal with branching structures. The non-deterministic behaviours introduced by DRT-like models are not well understood in the context of resource sharing.

Real-Time Multiprocessor Scheduling. In contrast to uniprocessor systems, scheduling real-time workloads on parallel architectures is a much harder challenge. There are various scheduling strategies, e.g., global and partition-based scheduling with mostly sufficient conditions for schedulability tests. A comprehensive survey on this topic is found in [21]. The introduction of multicore platforms adds another dimension of complexity due to on-chip resource contention. The known techniques for multiprocessor scheduling all rely on a safe WCET bound of tasks under idealized assumptions on the underlying platform. For multicore platforms, without proper isolation, it seems impossible to achieve

WCET bounds for tasks running in parallel on different processor cores. To the best of our knowledge, there is no work on bridging WCET analysis and multiprocessor scheduling.

Mixed-Criticality Systems. Integrating applications of different levels of criticality on the same processor chip is attracting increasing interest. Scheduling mixed-criticality workloads on uniprocessors has been studied intensively in recent years [14, 15, 24]. For multiprocessor systems, it is still an open area for research; a seminal work is found in [35]. Due to their potentially high computing capacity, we believe that multicore processors are more adequate for such mixed types of applications. Typically, a mixed criticality system operates in different modes. In the normal mode, the computation power of multicores may be used to improve the average performance of low-criticality applications and in case of exceptions, e.g., a timing error occurs, high-criticality applications should be prioritized. However, it is not well understood how to isolate and avoid interferences among the applications without fully partitioning the on-chip resources.

Further extensions of workload models. One may add new features or high-level structures on the existing models for expressiveness and abstraction purposes. We see two interesting directions:

Fork-Join Real-Time (FJRT) tasks: Recently, an extension of the DRT task model to incorporate fork/join structures is proposed by Stigge et al. Instead of following just *one* path through the graph, the behavior of a task includes the possibility of *forking* into different paths at certain nodes, and *joining* these paths later. Syntactically, this is represented using hyperedges. A hypergraph generalizes the notion of a graph by extending the concept of an edge between two vertices to hyperedges between two sets of vertices. For details, we refer to [47]. The model is illustrated with an example in Figure 11. For the time of writing this survey, no efficient analysis method for FJRT is known.

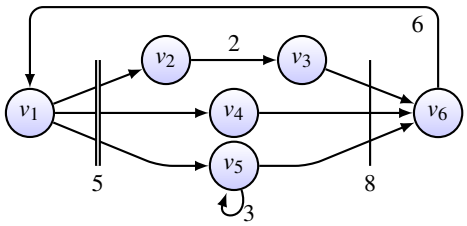


Fig. 11. Example FJRT task. The fork edge is depicted with an intersecting double line, the join edge with an intersecting single line. All edges are annotated with minimum inter-release delays $p(U,V)$. The vertex labels are omitted in this example. Assuming that vertex v_i releases a job of type J_i, a possible job sequence containing jobs with their types and absolute release times is $\sigma = [(J_1,0),(J_2,5),(J_5,5),(J_4,6),(J_3,7),(J_5,8),(J_6,16),(J_1,22)]$.

Mode Switches: A system may operate in different modes demonstrating different timing behaviours. We consider a simple model using general directed graphs where

the nodes stand for modes, assigned with a set of tasks to be executed in the corresponding mode, and edges for mode switches that may be triggered by an internal or external event and guarded by a timing constraint such as a minimal separation distance. Mode switching is a concept that has been studied in different contexts. Many protocols for mode switches have been proposed [41] to specify what should be done during the switch. In a recent work, the authors [25] show that a mixed criticality task system can be modeled nicely using a chain of modes representing the criticality levels where mode switches are triggered by a task over-run that may occur at any time, and on each mode switch, tasks of lower-level criticality will be dropped and only tasks of higher-level criticality are scheduled to run according to their new task parameters in the new mode. The authors present a technique for scheduling the mixed criticality workload described in directed acyclic graphs. An interesting direction for future work is scheduling of mode switches in general directed graphs, which involves fixed-point computation due to cyclic structures.

Tools for schedulability analysis. Over the years, many models and analysis techniques have been developed. It is desirable to have a software tool that as input takes a workload description in some of the models and a scheduling policy and determines the schedulability. A tool for task automata has been developed using techniques for model checking timed automata [2]. Due to the analysis complexity of timed automata, it suffers from the state-explosion problem. For the tractable models including DRT in the hierarchy of Figure 1, a tool for schedulability analysis is currently under development in Uppsala based on the path abstraction technique of [46].

References

1. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES - A Tool for Modelling and Implementation of Embedded Systems. In *Proc. of TACAS 2002*, pages 460–464. Springer-Verlag, 2002.
3. N.C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Proc. of RTOSS 1991*, pages 133–137, 1991.
4. Neil C Audsley, Alan Burns, Robert I Davis, Ken W Tindell, and Andy J Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2):173–198, 1995.
5. T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, April 1991.
6. Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Syst.*, 17(1):5–22, 1999.
7. Sanjoy Baruah and Joël Goossens. Scheduling real-time tasks: Algorithms and complexity. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 3, 2004.
8. Sanjoy K. Baruah, , A.K. Mok, and L.E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. of RTSS 1990*, pages 182–190, 1990.
9. Sanjoy K. Baruah. A general model for recurring real-time tasks. In *Proc. of RTSS*, pages 114–122, 1998.
10. Sanjoy K. Baruah. Feasibility Analysis of Recurring Branching Tasks. In *Proc. of EWRTS*, pages 138–145, 1998.

11. Sanjoy K. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. *Real-Time Systems*, 24(1):93–128, 2003.
12. Sanjoy K. Baruah. Preemptive Uniprocessor Scheduling of Non-cyclic GMF Task Systems. In *Proc. of RTCSA 2010*, pages 195–202, 2010.
13. Sanjoy K. Baruah. The Non-cyclic Recurring Real-Time Task Model. In *Proc. of RTSS 2010*, pages 173–182, 2010.
14. Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Trans. Computers*, 61(8):1140–1152, 2012.
15. Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. Response-time analysis for mixed criticality systems. In *RTSS*, pages 34–43, 2011.
16. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124, 2003.
17. Vandy Berten and Joël Goossens. Sufficient FTP Schedulability Test for the Non-Cyclic Generalized Multiframe Task Model. *CoRR*, abs/1110.5793, 2011.
18. J.Y.L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science. Springer, 2001.
19. G.C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Realtime Systems. Springer, 2011.
20. Samarjit Chakraborty, Thomas Erlebach, and Lothar Thiele. On the complexity of scheduling conditional real-time code. In *Proc. of WADS 2001*, pages 38–49, 2001.
21. Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.
22. Friedrich Eisenbrand and Thomas Rothvoß. Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In *Proc. of RTSS 2008*, pages 397–406, 2008.
23. Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proc. of SODA 2010*, pages 1029–1034, 2010.
24. Pontus Ekberg and Wang Yi. Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, pages 135–144, 2012.
25. Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Norwell, MA, USA, 2013. Kluwer Academic Publishers.
26. Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.
27. Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.
28. Elena Fersman, Paul Pettersson, and Wang Yi. Timed Automata with Asynchronous Processes: Schedulability and Decidability. In *Proc. of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.
29. Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Resource Sharing Protocols for Real-Time Task Graph Systems. In *Proc. of ECRTS 2011*, pages 272–281, 2011.
30. C.-C. J. Han. A Better Polynomial-Time Schedulability Test for Real-Time Multiframe Tasks. In *Proc. of RTSS*, pages 104–, Washington, DC, USA, 1998. IEEE Computer Society.
31. Mathai Joseph and Paritosh K. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29:390–395, 1986.
32. John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proc. of RTSS*, pages 261–270, 1987.
33. Joseph Y.-T. Leung and M.L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115 – 118, 1980.
34. Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.

35. Haohan Li and Sanjoy K. Baruah. Outstanding paper award: Global mixed-criticality scheduling on multiprocessors. In *ECRTS*, pages 166–175, 2012.
36. C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, 1973.
37. Wan-Chen Lu, Kwei-Jay Lin, Hsin-Wen Wei, and Wei-Kuan Shih. New Schedulability Conditions for Real-Time Multiframe Tasks. In *Proc. of ECRTS*, pages 39–50, 2007.
38. A. K. Mok. Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment. Technical report, Cambridge, MA, USA, 1983. Ph.D. thesis.
39. Aloysius K. Mok and Deji Chen. A General Model for Real-Time Tasks. Technical report, 1996.
40. Aloysius K. Mok and Deji Chen. A Multiframe Model for Real-Time Tasks. *IEEE Trans. Softw. Eng.*, 23(10):635–645, 1997.
41. L.T.X. Phan, Insup Lee, and O. Sokolsky. A semantic framework for mode change protocols. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 91 –100, april 2011.
42. Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, Technical University of Braunschweig, Braunschweig, Germany, 2004.
43. L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175 –1185, sep 1990.
44. Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004.
45. Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the Tractability of Digraph-Based Task Models. In *Proc. of ECRTS 2011*, pages 162–171, 2011.
46. Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The Digraph Real-Time Task Model. In *Proc. of RTAS 2011*, pages 71–80, 2011.
47. Martin Stigge, Pontus Ekberg, and Wang Yi. The Fork-Join Real-Time Task Model. In *Proc. of ACM SIGBED Review*, 2012. To appear.
48. Martin Stigge and Wang Yi. Hardness Results for Static Priority Real-Time Scheduling. In *Proc. of ECRTS 2012*, pages 189–198, 2012.
49. Hiroaki Takada and Ken Sakamura. Schedulability of Generalized Multiframe Task Sets under Static Priority Assignment. In *Proc. of RTCSA 1997*, pages 80–86, 1997.
50. Noel Tchidjo Moyo, Eric Nicollet, Frederic Lafaye, and Christophe Moy. On Schedulability Analysis of Non-cyclic Generalized Multiframe Tasks. In *ECRTS 2010*, pages 271–278, 2010.
51. L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS 2000*, volume 4, 2000.
52. Tei wei Kuo, Li pin Chang, Yu hua Liu, and Kwei jay Lin. Efficient online schedulability tests for real-time systems. *IEEE Transactions On Software Engineering*, 29:734–751, 2003.
53. A. Zuhily and A. Burns. Exact Scheduling Analysis of Non-Accumulatively Monotonic Multiframe Tasks. *Real-Time Systems Journal*, 43:119–146, 2009.