Time Petri Nets for Schedulability Analysis of the Transactional Event Handlers of P-FRP

Chaitanya Belwal, Albert M.K. Cheng, Yuanfeng Wen

Computer Science Department
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

UH-CS-12-04
May 21, 2012

Keywords: Functional programming, Reactive programming, Schedulability analysis, Worst-case response Time, Formal methods, Time Petri Nets

Abstract

Priority-based FRP (P-FRP) is functional programming formalism for reactive systems that guarantees real-time response. Preempted tasks in P-FRP are aborted and have to restart when no higher priority tasks are present in the execution queue. The abort and eventual restart makes the response time of a lower priority task completely dependent on the execution pattern of higher priority tasks. Exact schedulability analysis methods of P-FRP that have been presented so far require the evaluation of all release scenarios of higher priority tasks. Unfortunately, the number of such scenarios scale exponentially with the size of the task set making exact schedulability analysis of this execution model a computationally expensive proposition. The formal method of Time Petri Net (TPN) has previously been used for schedulability analysis of preemptive and non-preemptive models. TPNs for P-FRP or other transaction like execution models have not been developed yet. In this paper, we develop TPN models for the transactional execution model of P-FRP and show that TPNs offer an efficient alternative for schedulability analysis of this model. We have implemented our TPNs in the model checker ROMEO and have validated the correctness of our models through experimental task sets.

* This work is supported in part by U.S. National Science Foundation under Award no. 0720856

Time Petri Nets for Schedulability Analysis of the Transactional Event Handlers of P-FRP

Chaitanya Belwal, Albert M.K. Cheng and Yuanfeng Wen
Department of Computer Science,
University of Houston, TX, USA
{cbelwal, cheng, wyf}@cs.uh.edu

Abstract

Priority-based FRP (P-FRP) is functional programming formalism for reactive systems that guarantees real-time response. Preempted tasks in P-FRP are aborted and have to restart when no higher priority tasks are present in the execution queue. The abort and eventual restart makes the response time of a lower priority task completely dependent on the execution pattern of higher priority tasks. Exact schedulability analysis methods of P-FRP that have been presented so far require the evaluation of all release scenarios of higher priority tasks. Unfortunately, the number of such scenarios scale exponentially with the size of the task set making exact schedulability analysis of this execution model a computationally expensive proposition. The formal method of Time Petri Net (TPN) has previously been used for schedulability analysis of preemptive and non-preemptive models. TPNs for P-FRP or other transaction like execution models have not been developed yet. In this paper, we develop TPN models for the transactional execution model of P-FRP and show that TPNs offer an efficient alternative for schedulability analysis of this model. We have implemented our TPNs in the model checker ROMEO and have validated the correctness of our models through experimental task sets.

Index Terms

Functional programming, Reactive programming, Schedulability analysis, Worst-case response Time, Formal methods, Time Petri Nets

1. Introduction

Functional Reactive Programming (FRP) [28] is a declarative programming language for modeling and implementing reactive systems. It has been used for a wide range of applications, notably, graphics [7], robotics [18], and vision [19]. FRP elegantly captures continuous and discrete aspects of a hybrid system using the notions of *behavior* and *event*, respectively. Because this language is developed as an embedded language in Haskell [12], it benefits from the wealth of abstractions provided in this language. However, Haskell provides no real-time guarantees and therefore, neither does FRP.

To address this limitation, resource-bounded variants of FRP were studied ([26],[27],[28]). Recently, it was shown that a variant called priority-based FRP (P-FRP) [14], combines both the semantic properties for FRP, guarantees resource boundedness, and supports assigning different priorities to different events. In P-FRP, higher priority events can preempt lower-priority ones. However to maintain guarantees of type safety and state-less execution, the functional programming paradigm requires the execution of a function to be atomic in nature. To comply with this requirement as well as allow preemption of lower priority events, P-FRP implements a transactional model of execution. Using only a copy of the state during event execution and atomically committing these changes at the end of the event handler (or *task*), P-FRP ensures that handling an event is an "all or nothing" proposition. This preserves the easily understandable semantics of the FRP and provides a programming model where response times to different events can be tweaked by the programmer, without ever af-

fecting the semantic soundness of the program. Thus, a clear separation between semantics of the program and the responsiveness of the implementation of each handler is achieved.

Prior work in P-FRP has dealt with computing approximate ([14], [21]) response time bounds as well as methods to determine response time under a given release scenario ([2], [3]). Multi-processor scheduling of P-FRP has been studied in [22]. For exact schedulability analysis, computing the Worst-Case Response Time (WCRT) of a task is essential. In the preemptive model of execution, a critical instance of release leading to the WCRT exists [15]. However as shown in [2], no such critical instant exists in P-FRP and the WCRT can be different for each unique task set. The only known method of computing the exact WCRT for P-FRP requires evaluation of all release scenarios within a lower and upper bound of release offsets of higher priority P-FRP tasks. Unfortunately, this technique scales exponentially with the size of the task set and is impractical even for small sized task sets.

In a seminal paper, Merlin and Farber [16] introduced Time Petri Net (TPN) as a formal method for verification of timed systems. TPN was shown to be more expressive than Timed Petri Net (TdPN) which was developed by Ramachandani [20]. Following Merlin and Farber's paper, several researchers have developed techniques which apply TPN for schedulability analysis of real-time systems, notable among these being the works of [24], [11] and [29]. These available methods only deal with different forms of preemptive and non-preemptive execution and so far the application of TPN to a transactional execution model like P-FRP has not been studied.

The use of TPN for schedulability analysis in an execution model like P-FRP's, provides more immediate benefits than for the standard preemptive or non-preemptive. This is because while several polynomial time exact schedulability analysis techniques of the preemptive execution model are available, no polynomial time methods exist for such analysis in P-FRP. Using expressive formalism like TPNs to model the execution of P-FRP and analyzing the schedulability of the model allows us to leverage highly efficient state space exploration techniques available in software tools. Hence, TPN offers an efficient alternative for schedulability analysis in P-FRP.

In this paper, we present Time Petri Net models that can be used for schedulability analysis in P-FRP. These models have been analyzed in a publicly available TPN tool called ROMEO [10]. The results from our TPN model have been validated against simulations using experimental task sets.

After presenting the notations and execution model of P-FRP (Section 2), we present a primer on Time Petri Nets and ROMEO (Section 3). We introduce important concepts of our TPN design through a simple TPN for 2-task sets (Section 4). Using the TPN for 2-task set we show how to build a TPN for 3-tasks (Section 5), followed by which we present the methodology to design a TPN for any general P-FRP *n*-Task set (Section 6). We discuss the results of our experimental analysis (Section 7) followed by which we conclude with a review of related work (Section 8) and a reflection on our results (Section 9).

There are several other models that utilize some form of transactional execution. Notable among these are the transactional memory systems [13] and lock-free execution of critical sections [1]. The development of these systems was primarily motivated by the need to avoid concurrency or precedence constraint issues, which have been a problem in the classical preemptive model [23]. Though the execution semantics of these models are different, TPN models developed in this paper also serve as a valuable resource for schedulability analysis of these execution models.

2. Execution Model and Notations

In this section, we introduce the notations used in the rest of the paper. In addition, we review the P-FRP execution model and assumptions made in this study.

2.1 Notations

The notations and formal definitions used in the paper are as follows:

- Let task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n sporadic tasks. Γ_n is also called a ***n*-task set**.
- If $i > j$ then τ_i has a higher priority than τ_j, a convention used in Linux and Windows operating systems.

- $\tau_{i,k}$ represents the k^{th} job of τ_i.
- T_j is the **minimum arrival time separation** between two successive jobs of τ_j.
- C_j is the **worst-case execution time (WCET)** for τ_j.
- D_j is the **relative deadline** of τ_j. In this paper, we assume $D_j = T_j$.
- **Interference** on τ_k is the action where the execution of τ_k is interrupted by the release of a higher priority task.

We now define the execution model and assumptions made in our study.

2.2 Execution Model and Assumptions

In this study, all tasks are assumed to execute in a uniprocessor system with no precedence constraints. Events which trigger the tasks are sporadic and are converted to a periodic model by assuming minimum time interval between any two events of the same type. When a job of a higher priority task τ_i is released, it can immediately preempt an executing lower priority task and changes made by the lower priority task are rolled back. The lower priority task will be restarted when the higher priority task has completed execution. Due to P-FRP's transactional nature of execution, all tasks are assumed to run without concurrency constraints. In the algorithms to derive the actual response time of a task τ_j, we have considered the release offset of τ_j to be 0.

When a task is released, it enters an execution queue which is arranged by priority order such that all arriving higher priority tasks are moved to the head of the queue. The length of the queue is bounded, and no two instances of the same task can be present in the queue at the same time. This requires a task to complete execution before the release of its next job. To maintain this requirement we assume a *hard* real-time system with task deadline equal to the time period between jobs. Hence,

$\forall \tau_k \in \Gamma_n, D_k = T_k$.

A task set is schedulable in some time interval only if no task in the set has a deadline miss. While TPNs are capable of building dense time models, in this work we have only considered modeling in discrete time.

Once τ_i enters the execution queue, two situations are possible. If a task of a priority lower than *i* is executing, it will be immediately preempted and τ_i will start execution. If a task having a higher priority than *i* is executing, then τ_i will wait in the queue and start execution only after the higher priority task has completed. When a task starts execution it creates a temporary copy of the state (represented by variables in register/memory) and upon completion it overwrites the original state with the temporary copy in an atomic operation. During these state copy operations, all system interrupts are disabled and no task is allowed to enter the execution queue.

2.3 Preemptive vs. P-FRP Execution Model

The preemptive execution model has been the focus of real-time research over the past decade. Several state-of-the-art methods including those using Time Petri Nets are available for analysis of this model. Since our readers may be more familiar with the preemptive execution in this section, we highlight the difference in execution semantics between the P-FRP and preemptive execution models.

Consider the 3-Task set:

Task	C_i	T_i
τ_1	4	36
τ_2	4	15
τ_3	3	10

If all tasks are released together, τ_1 completes at time 14 in preemptive execution (*fig. 1(a)*). If tasks are executed in P-FRP, τ_1 completes at time 27 (*fig. 1(b)*). The response time of τ_1 is higher due to the abort of τ_1 at times 10, 15 and 20. It can also be seen from *fig. 1(b)*, that even though the WCET of τ_1 is 4, it executes for a total of 10 time units. Hence, in P-FRP the actual processor time that a task takes to complete execution cannot be determined solely by using the information available *a priori*. Note that we have not considered any data

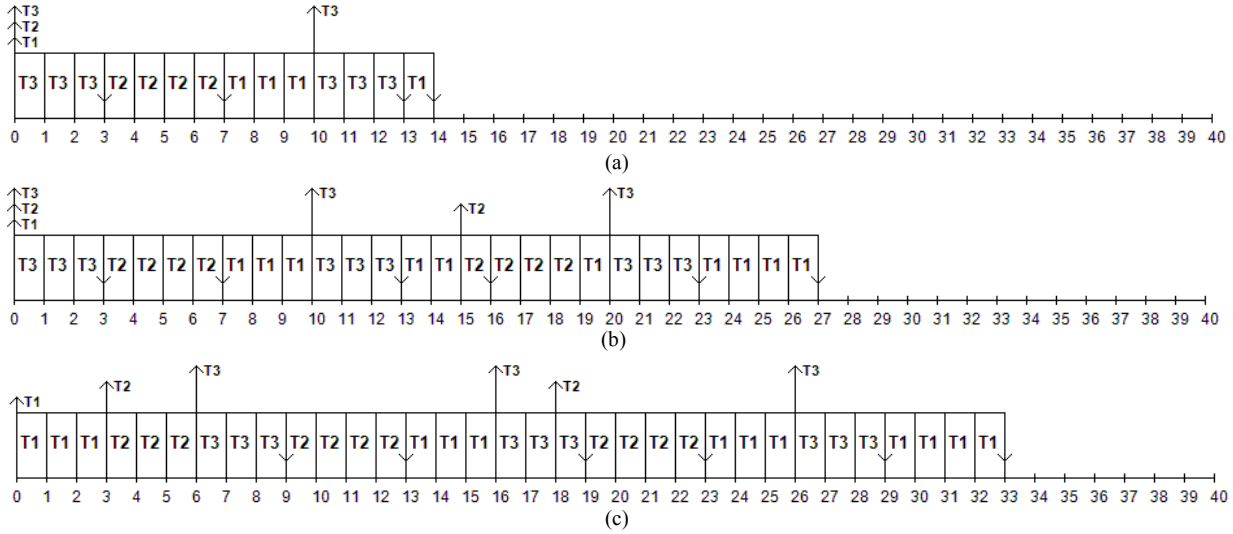

Figure 1. Task graph highlighting execution semantics (a) Preemptive execution when all tasks are released at the same time (b) P-FRP execution when all tasks are released at the same time (c) P-FRP execution in the worst-case release scenario

sharing between tasks which require use of locks or other concurrency control mechanism. With data sharing some tasks can experience blocking in which case the response time of τ_1 can be different in preemptive execution. Data sharing between tasks has no effect on the execution semantics of P-FRP since, every task executes atomically.

In their seminal paper, Liu and Layland showed the worst-case response time (WCRT) of a task occurs when all tasks are released at the same time. Therefore in the preemptive model, the WCRT of τ_1 is also 14, and *fig. 1(a)* shows the worst-case task execution of our sample task set. However in P-FRP, such a release is not guaranteed to the worst-case. The WCRT of τ_1 is 33 and occurs when τ_2, τ_3 are released at times 3 and 6 respectively (*fig. 1(c)*). This WCRT was determined by evaluating all possible release scenarios of tasks τ_2 and τ_1 in the interval [0,45). Such a method of determining WCRT is computationally intensive and time consuming and alternate techniques are required.

Time Petri Nets offer an alternate mechanism for schedulability analysis of P-FRP.

3. Time Petri Nets

Merlin and Farber [16] presented Time Petri Net as an extension of classical Petri nets where the firing time of transitions are bounded in time. For details on classical Petri Net models readers can refer to [17]. The formal definition of a Time Petri Net is given as follows:

A Time Petri Net is a tuple (P, T, B, F, M_O, SI) where:

- $P = \{p_1, p_2, p_3, \ldots, p_n\}$ is a finite non-empty set of places.
- $T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a finite nonempty, set of transitions.
- B: $P \times T \rightarrow N$ is the **backward incidence** function, where $\boldsymbol{N}$ is the set of non-negative integers.
- F: $T \times P \rightarrow N$ is the **forward incidence** function.
- M_O is the **initial marking**.
  (P, T, B, F and M_O together define a Petri net).
- SI is a mapping called **static interval**, $\forall t \in T$, $SI(t) = [SEFT(t), SLFT(t)]$, where $SEFT(t)$ is the **static earliest firing time** and $SLFT(t)$ the **static latest firing time**.

The **state** of a TPN is a pair where $S = (M, I)$:

- M is a marking.

- I is a firing interval set which is a vector of possible firing times. The number of entries in this vector is given by the number of the transitions enabled by marking M.

3.1 TPN modeling using ROMEO

Developed by IRCCyN in France, ROMEO [10] is a tool that allows for the description and evaluation of Time Petri Net models. Using a Graphical User Interface (GUI), places and transitions can be drawn and connected by directional arcs. The static earliest and latest firing times are added to the transitions using an user entry form. Markings are represented by placing tokens in places. In ROMEO, each place has a unique numerical index associated with it.

Once a TPN model is built, a separate simulation program is activated to simulate and analyze the model. ROMEO allows for interactive simulation where the user fires a ready transition through a mouse click. The tokens are dynamically moved to the next place in the forward incidence function after firing the transition. Reachability of states is analyzed using Timed Computational Tree Logic (TCTL) formatted queries which allow for checking the marking of a state in a specified time bound. Reachability analysis in TPN is done by converted a model to it state space graph.

The benefit of using tools like ROMEO is that we can leverage efficient algorithms for state space search that are implemented in these tools. For example, ROMEO implements the Differential Bound Matrices (DBM) technique which is an efficient and scalable method for state space search. By building a schedulability model of P-FRP in TPN, the schedulability problem is reduced to a reachability analysis problem which can be done in polynomial or pseudo-polynomial through methods like DBM. This allows for efficient schedulability analysis of P-FRP tasks, irrespective of the size of the task set.

We now demonstrate a simple TPN schedulability model for a P-FRP task set containing 2 tasks.

4. TPN for 2-Task Sets

In this section, we present a TPN model for P-FRP 2-task sets. Important concepts dealing with our TPN design can be explained with this simple case, and the model is expanded to include multiple tasks in subsequent sections. To design our TPN, we use the following task set as a case study:

Task	C_i	T_i
τ_1	4	12
τ_2	3	10

The TPN derived for this task set can be used for the schedulability analysis of any 2-task set by changing certain parameters defined in the TPN.

As discussed previously, there is no known single instance of release of higher priority tasks that can lead to the WCRT of a lower priority P-FRP task. Determination of the WCRT requires analysis of all release orderings of higher priority task for their response times. Such a scenario can be modeled quite well in TPN by specifying the $SEFT(t)$ and $SLFT(t)$ values for transition t to bound the release time of higher priority tasks. The first job of higher priority tasks are released non-deterministically in the specified bounds. In this paper, the general convention of ROMEO prefixing a place name by 'P' and a transition by 'T' has been followed. However to clarify the models, we have changed certain place names to denote their functionality.

We now look at the TPN design that models the periodic releases of tasks.

4.1 Task Release TPN

An important consideration in schedulability analysis of real-time systems, is the release of multiple jobs of higher priority tasks which can interfere with the execution of lower priority tasks. In TPN, we model the peri-

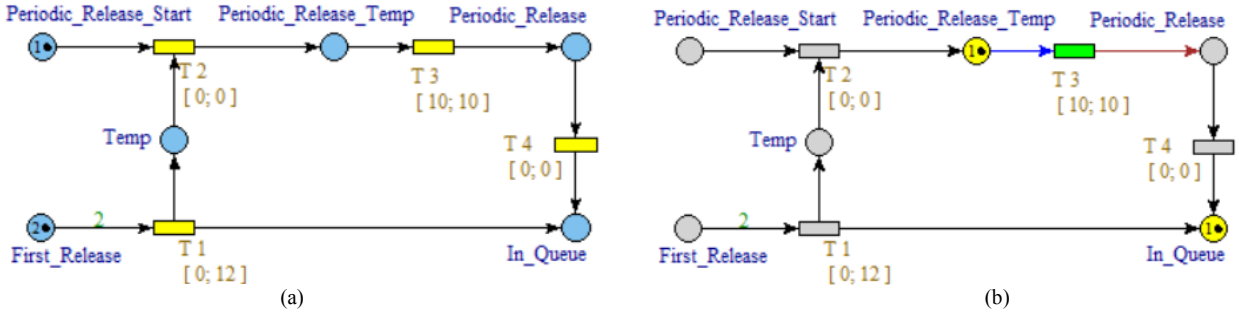

Figure 2. TPN for periodic task release. This TPN model was built using the GUI drawing tool of ROMEO. (a) Initial markings of places (b) Markings after the 1st job is present in the execution queue and the periodic job is ready to be released after firing 'T3'

odic release by introducing tokens in initial places. Each token represents a single job of the task. First the maximum number of job releases of a higher priority task τ_i is computed using the following formula:

Number of jobs of $\tau_i = \left\lceil \frac{D_j}{T_i} \right\rceil$ where τ_j is the task whose response time has to be determined.

This is the maximum number of tokens that need to be placed since if τ_j does not complete execution by time D_j, it is deemed unschedulable. *Fig. 2(a)* shows the TPN design for the release of jobs of τ_2 of our sample 2-task set. The deadline of τ_1 is 12 hence, the maximum number of jobs of τ_2 that can be released within this time is $\left\lceil \frac{12}{10} \right\rceil = 2$, including the 1st job.

The release of the 1st job of a task is considered separately since it can be released at any time in the interval [0, D_j). Subsequent task releases have to follow the periodic order. 2 tokens are placed for the 1st job while $\left\lceil \frac{D_j}{T_i} \right\rceil$ – 1 tokens are placed for periodic releases. In *fig. 2*(*a*), 2 tokens are in the place 'First_Release' while 1 token is in the place 'Periodic_Release_Start'. Transition 'T1' can non-deterministically fire anytime in the time interval [0, 12). This semantics is represent by setting *SEFT*(*T1*)=0 and *SLFT*(*T1*)=12. Since the weight of the arc from place 'First_Release' is 2, after 'T1' is fired 1 token is moved to place 'In_Queue' while 1 token moves to the place 'Temp' and waits for 'T2' to fire. A token present in the place 'In_Queue' denotes a job of τ_2 is present in the execution queue. The token in 'In_Queue' is used by the scheduler TPN defined in subsequent sections.

As *SEFT*(*T2*)=0 and *SLFT*(*T2*)=0, 'T2' is immediately fired once a token is available in 'Temp'. After 'T2' is fired, all tokens from 'Periodic_Release_Start' are moved to 'Periodic_Release_Temp'. The weight of the arcs connected these two places has a weight equal to the number of tokens in 'Periodic_Release_Start'. The values *SEFT*(*T3*)=10 and *SLFT*(*T3*)=10 allow 'T3' to be fired after 10 time units which is the arrival time period between jobs of τ_2. The weight of the arcs connected from 'T3' is 1 hence after time intervals of 10, a single token will be moved from 'Periodic_Release_Temp' to 'Periodic_Release'. Once in 'Periodic_Release' the token reaches 'In_Queue' without any time delay since *SEFT*(*T4*)=0 and *SLFT*(*T4*) = 0. This way after the release of the 1st job of τ_2 in the interval [0, D_1), rest of the jobs are released at periodic time intervals of T_2.

The markings in the TPN after firing 'T1' releases a token in the execution queue is seen in *fig.2*(*b*). The token in 'Periodic_Release_Temp' will reach 'Periodic_Release' when 'T3' is fired after 10 time units.

4.2 Scheduler TPN

In this section, we derive the scheduler TPN for a 2-task set which utilizes the task release TPN introduced earlier. Before that we introduce the important definition of abort costs.

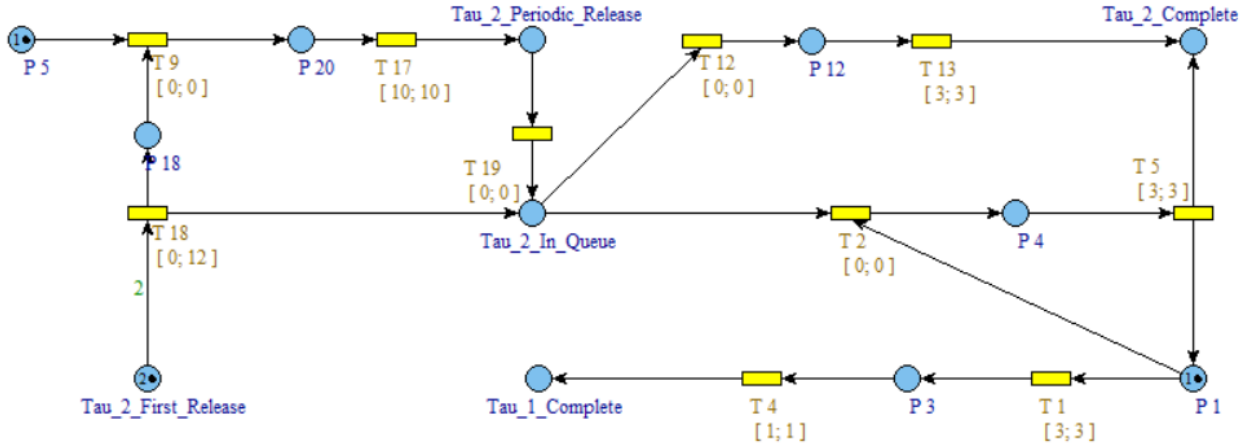

Figure 3. Time Petri Net model for schedulability analysis of a 2-task P-FRP system.

Definition. In the P-FRP execution model, preempted tasks are aborted. The amount of time spent in aborted execution of a task τ_j is called the **abort cost**. When τ_j is preempted immediately after starting, a minimum abort cost of 1 time unit in added to its response time. If τ_j is preempted just before it is about to complete execution, maximum abort costs of $C_j - 1$ time units are added to its response time.

Fig. 3 shows the TPN scheduler model of our 2-task P-FRP set. The task release TPN for τ_2 has been included and when a token is available in the place 'Tau_2_In_Queue' it implies that a job of this higher priority task is available for execution. The execution of τ_1 which is assumed to be released at time 0, is represented by places 'P1', 'P3' and 'Tau_1_Complete'. A single token to denote the execution state of τ_1 is placed in 'P1' and if the token reaches 'Tau_1_Complete' it implies that τ_1 has completed execution. The maximum abort cost that can be induced on τ_1 is (4 – 1) = 3, hence τ_1 can only be preempted if it has executed for any time in the interval (0,3]. This semantics is reflected by setting *SEFT*(T1) = 0 and *SLFT*(T1) = 3. After 'T1' fires, τ_1 enters place 'P3' and waits for 'T3' to fire. *SEFT*(T3) = 1 and *SLFT*(T3) =1 which allows 'T3' to fire after exactly 1 time unit allowing the marking to reach the place 'Tau_1_Complete'.

While τ_1 is executing (tokens in place 'P1'), it can be preempted by arrival of jobs of τ_2. The preemption of τ_2 is denoted by the transition 'T2' which can immediately fire (*SEFT*(T2)=*SLFT*(T2) = 0) once tokens are available in 'P1' and 'Tau_2_In_Queue'. After 'T2' is fired 2 tokens reach 'P4' and wait for 'T5' to fire. *SEFT*(T5) and *SLFT*(T5) values are set to the execution time of τ_2. Note that no other transitions are possible from 'P4' since τ_2 cannot be preempted by any other task. After 'T5' is fired, a token is returned to 'P1' and another token reaches 'Tau_2_Complete'. The number of tokens in 'Tau_2_Complete' denotes the number of jobs of τ_2 that have completed execution. Once a token reaches 'P1' it waits for 'T1' to fire denoting that τ_1 has restarted execution. If another token reaches the place 'Tau_2_In_Queue' before 'T1' is fired, the similar process described before is repeated.

After τ_1 has completed, jobs of τ_2 can directly execute through transition 'T12', which only requires a single token to be present in 'Tau_2_In_Queue'. After 'T12' is fired, a token reaches 'P12' where after 3 time units it reaches 'Tau_2_Complete'. This path is not of particular interest since once the token in 'P1' reaches 'Tau_1_Complete', it implies that τ_1 has completed execution. However for completeness of our TPN, it is required to model the case when no job of τ_1 is ready for execution.

In our TPN model, it is clear that if jobs of τ_2 are released at a rate which does not allow transition 'T1' to fire, τ_1 will never be able to complete execution. We now show how the TPN for 2-task sets can be used to determine schedulability.

4.3 Schedulability Analysis

To determine the schedulability of a 2-Task set we have to determine if lower priority task τ_1 can complete execution before its deadline. In our TPN model this problem is translated to the place 'Tau_1_Complete' getting marked with a token within bounded time. Or, we have to check if any release scenario of higher priority task τ_2 exists in which the token present in 'P1' does not reach 'Tau_1_Complete' after the deadline of τ_1 has passed.

ROMEO allows such queries on a TPN model using the syntax of Time Computational Tree Logic (TCTL) which is derived from the original syntax of Computational Tree Logic (CTL). Readers can refer to [8] for a syntactical reference on CTL. The TPN model defined by the user is converted to a state space graph. Every possible state that can be reached using the markings and time values defined in the TPN is contained in the graph. Efficient state space search techniques like DBM are used to find specific states defined in the TCTL query. Details on implementation of TCTL in ROMEO are given in [6]. The ROMEO TCTL query on the TPN model of *fig. 3* for determining schedulability of τ_1 in our sample task set is:

EF[*13,13*](*M*(*22*) = *0*).

The query returns a Boolean *true* or *false* depending whether it is satisfied or not. The query specifies that starting at time 13 and ending at time 13, is it possible for the place at index number 22 (index for place 'Tau_1_Complete') not to be marked with a single token ? If *true*, then there exists a scenario in which the token at 'P1' is unable to reach 'Tau_1_Complete' within the given time bound. Since, the deadline of τ_1 is 12, if 'Tau_1_Complete' is unmarked after this time it implies that there exists a worst-case release scenario of τ_2 in which τ_1 misses its deadline and the task set is unschedulable. Hence, for τ_1 to be schedulable in all release scenarios the query *EF*[*13,13*](*M*(*22*)=*0*) should return a Boolean value of *false*.

4.4 Generality

The TPN model of *fig. 3*, can be used for schedulability analysis of any general P-FRP 2-task set. For a task set $\Gamma_2 = \{\tau_1, \tau_2\}$, all that has to be done is change *SEFT* and *SLFT* values for various transactions as described below:

- *SLFT*(T18) = T_1.
- *SEFT*(T17) = *SLFT*(T17) = T_2.
- *SEFT*(T5) = *SLFT*(T5) = C_2.
- *SEFT*(T13) = *SLFT*(T13) = C_2.
- *SEFT*(T1) = *SLFT*(T1) = $C_1 - 1$.

The weight of arcs connected to 'T9' and the number of tokens in 'P5' is set to: $\left\lceil \frac{D_1}{T_2} \right\rceil - 1$. The TCTL query to check the schedulability of Γ_2 will be *EF*[D_1+*1*, D_1+*1*](*M*(*22*)=*0*).

5. TPN for 3-Task Sets

We extend the 2-task TPN presented earlier, for schedulability analysis of a 3-task set. As before, we use a sample task set to illustrate the model of the TPN. The task set considered for the 3-task set is the same as considered in section 2.3.The TPN model for this task set is given in *fig. 4*.

Places 'Tau_3_First_Release', 'P18', 'P5', 'P20', 'Tau_3_Periodic_Release', 'Tau_3_In_Queue' deal with the first and periodic release of jobs of τ_3. *SEFT* and *SLFT* values of transitions 'T18' and 'T17' are assigned relevant values. Places 'Tau_2_First_Release', 'P14', 'P15', 'P16', 'Tau_2_Periodic_Release', 'Tau_2_In_Queue' deal with the first and periodic release of jobs of τ_3, and *SEFT*, *SLFT* values of transitions

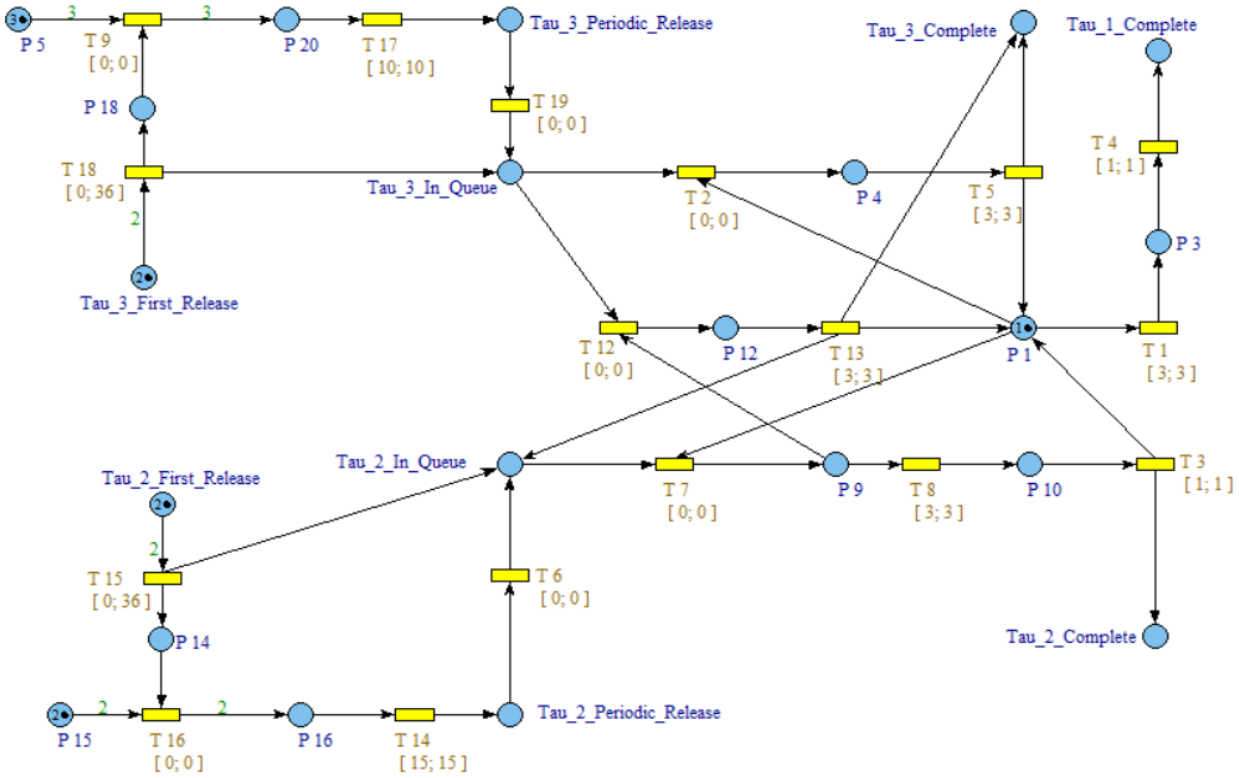

Figure 4. Time Petri Net model for schedulability analysis of a 3-task P-FRP system.

'T15' and 'T14' are assigned values accordingly. Places 'P1' and 'P3' deal with the execution of τ_1. As in 2-task sets, a single token is placed in 'P1' and when the token reaches 'Tau_1_Complete' it implies that τ_1 has completed execution.

$T_1 = 36$, hence the number of jobs of τ_2 that can be released till the deadline of τ_1 are, $\left\lceil \frac{36}{15} \right\rceil = 3$. Therefore, 2 tokens are placed in 'P15' which is part of the release TPN of τ_2. Similarly, $\left\lceil \frac{36}{10} \right\rceil - 1 = 3$ tokens are places in 'P5' for jobs of τ_3. Arcs from transitions 'T16' and 'T9' are assigned weights accordingly.

Once a job of τ_2 is released, a token is placed in 'Tau_2_In_Queue' and the token moves through places 'P9', 'P10' and finally, 'Tau_2_Complete'. 'P9' is reached after firing 'T7' which requires a token to be present in 'P1' and 'Task_2_In_Queue'. This way τ_1 is not allowed to execute once τ_2 commences execution. After 'T3' fires, τ_2 has completed execution and one token is returned to 'P1' while the other reaches 'Tau_2_Complete'.

An important consideration is that while τ_2 can preempt τ_1, it can itself be preempted by τ_3. This semantics is reflected by firing transition 'T12' once tokens are available in 'P9'. To fire, 'T12' also requires a token to be present in 'Tau_3_In_Queue'. Note that once 'T12' is fired, the token which was originally in 'P1' is now in 'P12'. After 'T13' is fired τ_3 completes execution and a token is returned to 'Tau_2_In_Queue' while the remaining tokens reach 'P1' and 'Tau_3_Complete'. *SEFT*(T13) = *SLFT*(T13) = 3, to denote the execution of τ_3. Like for τ_3 the execution of τ_2 is broken into two parts. *SEFT*(T8) and *SLFT*(T8) values are set to the maximum abort costs that can be induced on τ_2. Once τ_2 executes for its maximum abort cost, 'T8' is fired allowing a token to reach 'Tau_2_Complete' once 'T3' fires after a single time unit.

If τ_3 is released and no job of τ_2 is executing or ready to execute, then τ_3 preempts the execution of τ_1. This semantics is reflected in transitions 'T2' and 'T5'. 'T5' denotes the execution of τ_3 therefore, *SEFT*(T5) = *SLFT*(T5) = 3. After 'T5' fires, a single token is returned to 'P1' while the other reaches 'Tau_3_Complete'.

5.1 Schedulability Analysis

As for 2-task sets, schedulability analysis requires determining if τ_1 can complete execution before its deadline in all release scenarios. In ROMEO, this can be accomplished by the following TCTL query:

EF[*37,37*](*M*(*22*)=*0*).

The query specifies that at time 37 is it possible for place at index 22 (i.e. 'Tau_1_Complete') to have no tokens. If at time 37, 'Tau_1_Complete' has no token it implies that a release scenario of higher priority tasks exists in which τ_1 misses it deadline. Hence for τ_1 to be schedulable, the query *EF*[*37,37*](*M*(*22*)=*0*) should be a Boolean value of *false*.

In certain task sets it may be possible that even though τ_1 completes execution before its deadline, some jobs of τ_2 have a deadline miss. To ascertain the schedulability of the overall task set it is important to check for this condition as well. TCTL queries are powerful enough to allow us to check for these conditions in the same TPN. There are two ways in which this can be done. First, we can use the same query format as for τ_1. Note that the non-deterministic release of the 1st job of τ_3 in the interval [0,36) will also determine the WCRT of τ_2. Hence, we have to check if it is possible for the 1st job of τ_2 to not complete before its deadline. This is achieved by the following TCTL query:

EF[*16,16*](*M*(*6*)=*0*).

The place at index 6 is 'Tau_2_Complete. For τ_2 to be schedulable this query should return a Boolean *false*. Another query that can determine the schedulability of τ_2 is described below.

In the period [0, 16), the maximum number of tokens that can be present in the place 'Tau_2_Complete' is 1 to account for a single job of τ_2 that is released in the interval [0, 16). If this job of τ_2 is schedulable under all release scenarios of τ_3 in the time period [0, 16), only one job of τ_2 will be executing. Hence the sum of the number of tokens in 'P16' and 'Tau_2_Complete', will not be less than (3 – 1) = 2. This check can be accomplished by the following TCTL query:

EF[*0,16*](*M*(*16*)+*M*(*6*)<*2*).

This query specifies to look for a state between times 0 and 16, where the sum of tokens in state 'P16' (index 16) and 'Tau_2_Complete' (index 6) is less than 2. If such a state exists it means that in the time interval [0, 16) there exists a scenario when more than one job of τ_2 is executing or in the execution queue resulting a deadline miss. Hence, for τ_2 to be schedulable this query should also return a Boolean *false*.

5.2 Generality

The TPN model of *fig. 4* can be used for schedulability analysis for any general P-FRP 3-task set $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ by making the following changes:

- *SLFT*(T15) and *SLFT*(T18) = T_1.
- *SEFT*(T14) = *SLFT*(T14) = T_2.
- *SEFT*(T17) = *SLFT*(T17) = T_3.
- *SEFT*(T1) = *SLFT*(T1) = $C_1 - 1$.
- *SEFT*(T8) = *SLFT*(T8) = $C_2 - 1$.
- *SEFT*(T5) = *SLFT*(T5) = C_3.
- *SEFT*(T13) = *SLFT*(T13) = C_3.

The weight of arcs connected to 'T9' and the number of tokens in 'P5' is set to: $\left\lceil \frac{D_1}{T_3} \right\rceil - 1$, while the weights of arcs connected to 'T16' and the number of tokens in 'P15' is set to $\left\lceil \frac{D_1}{T_2} \right\rceil - 1$. The TCTL queries to verify the schedulability of τ_2 and τ_1 are:

$EF[D_2+1,D_2+1](M(6)=0)$ and
$EF[D_1+1, D_1+1](M(22)=0)$, respectively.

The query:
$EF[0, D_2+1](M(16)+M(6)<2)$,

can also be used to determine the schedulability of τ_2.

6. TPN for *n*-Task Sets

In previous sections, we have derived TPN model for 2-task and 3-task sets. It is clear that unique TPN's are required for task sets of different sizes. It is not feasible to present TPN models for every incremental task set size. However all the TPN models use the same design philosophy hence, constructing a TPN for any general P-FRP n-task set is straightforward using the 2-task and 3-task TPN models presented earlier as references.

The following steps are required to build a TPN model for any general *n*-task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$.

- A release TPN as shown in *fig. 2* is required for tasks τ_n through τ_2. Required number of tokens depending on the number of the jobs of that task in the interval $[0, T_1]$ are placed in the release TPN. Once a job has been released it is made available in the place 'Tau_*i*_In_Queue' for task τ_i.
- The execution of tasks τ_1 through τ_{n-1} is done in 2 transitions. For any task τ_k the *SEFT* and *SLFT* values of the 1st transition are set to its maximum abort costs $C_k - 1$, while the *SEFT* and S*LFT* values for the 2nd transition are set to 1. This can be seen in transitions 'T8', 'T3' for τ_2 and 'T1' and 'T4' for τ_1 in *fig.4*. The execution of the highest priority task τ_n is represented by a single transition.
- When a lower priority task τ_k is executing it can be preempted by a higher priority task. Hence, places which denote the execution of τ_k should have arcs connected to transitions which fire when a token of high priority task τ_i is available in the place 'Tau_*i*_In_Queue'. Such transitions have to be put in the TPN for all tasks which have higher priority than τ_k, for $1 \le k < n$. For example, in *fig. 4* transitions 'T2' and 'T7' are fired when jobs of τ_1 and τ_2 respectively, are available. When 'T2' and 'T7' are fired they consume the token available in 'P1'. Once the higher priority task has completed, the token which denotes the execution of τ_k has to be returned. In *fig. 4*, the token is returned to 'T1' after 'T5','T13' and 'T3' are fired.

6.1 Schedulability Analysis

After the complex task of converting the schedulability model to TPNs is done, the difficult problem of schedulability analysis is reduced to a simple TCTL query. The TCTL query is done on a state-space graph built internally by ROMEO, which has unique states representing all possible release scenarios of higher priority tasks. To determine if a task τ_k is schedulable the following TCTL query is sufficient:

$EF[D_k+1,D_k+1](M(idx_k)=0)$,

where idx_k represents the index of the place 'Tau_k_Complete'. This queries tells the system to search the entire state-space graph for a case where the number of tokens at state $M(idx_k)$ is 0, in the time range $[D_k+1,D_k+1]$. The query returns either a boolean *true* or *false*. If the query returns *true* then there exists a situation when the token does not reach $M(idx_k)$ in the time $[D_k+1,D_k+1]$. Since a token reaches $M(idx_k)$ only when τ_k has completed execution, a *true* answer implies that τ_k has not completed execution by time D_k+1 and hence is unschedulable.

While we have checked that τ_k is schedulable under all possible release scenarios, it is also important to check no other higher priority task has a deadline miss till τ_k has completed execution. Hence, we need to check that all jobs of a higher priority task $\tau_i:i > k$ are able to completed execution. This is achieved through the following TCTL query:

$$EF[0,D_k+1](M(idx_i) < \left\lceil \frac{D_k}{T_i} \right\rceil - 1),\ \tau_i:i > k.$$

Here idx_i represents the index of the place 'Tau_i_Complete'for a task $\tau_i:i > k$.

7. Experimental Results

We have tested our TPN models against the example task sets used in this paper. To validate our models more rigorously, we also tested our models against experimental task sets having 2, 3 and 4 tasks. 50 task sets of sizes varying between 2 and 4 tasks were synthetically generated. The execution times for these task sets were selected from the range [5, 15], while their arrival periods were selected from [30,100]. Each of the task sets were unique in the sense that at least one task is different between any two task tests. All task sets were generated using the TGSIMEx system and the values from which task execution times and arrival periods were selected were intentionally kept small to reduce the time taken to run the simulations.

For each of the task sets, the schedulability of the lowest priority task was determined by enumerating all release scenarios and then determining the WCRT for each scenario using the method developed in [2], again using TGSIMEx. If the WCRT of the lowest priority task was more than then the task set was deemed unschedulable. The time taken in determining the WCRT for every task set through TGSIMEx was noted.

We then modified the TPN model to determine the schedulability for each of the task sets (by manually changing *SEFT* and *SLFT* values of related transitions). The results from these two approaches were the same, confirming the accuracy of TPN. Since currently, ROMEO has no inbuilt system to measure the time taken to run the TCTL queries, we informally mention about the time comparisons between simulation and TPN for doing the schedulability analysis.

Simulations took an average of 1, 8 and 20 minutes to run the simulation of 2, 3 and 4-task sets respectively in an Intel dual-core machine with Microsoft Windows Vista. The queries on TPN models using ROMEO were also executed on the same machine. For all the task sets the query results were almost instantaneous, with a delay of 3-4 seconds for a few 4-task sets. This clearly shows the effectiveness of the optimized state space analysis techniques available for TPN's and available for use in tools like ROMEO.

8. Related Work

Related work includes schedulability techniques developed for P-FRP as well as using Time Petri Nets for schedulability analysis in other execution models. Apart from a minimal discussion on response time given by Kaibachev et al [14], an algorithm to compute approximate values of WCRT is given by Ras and Cheng [21]. Though the average approximation factor of the WCRT derived by this algorithm is unknown, for several task sets it fails to give a result. This algorithm has also been used for response time analysis of P-FRP in multiprocessor systems by Ras and Cheng [22]. Important contributions on schedulability analysis of P-FRP have also been made by Belwal and Cheng ([2], [3]).

One of the first papers that studies actual TPN models for schedulability analysis in real-time systems was presented by Tsai et al [24]. In this paper, methods to transform a schedulability model to Time Petri Nets are presented and the authors prove that schedulability analysis is reduced to a state reachability problem. Xu et al [29] do schedulability analysis using TPN by separating timing and behavioral properties. They present TPN models which allow for easy schedulability analysis. The Petri Net models presented in [24] and [29] are theoretical in nature since they have not been implemented in a software tool, and thus not been validated. Furfaro and Negro [9] also have present Time Petri Models for schedulability analysis. In [9], the TPN models have been translated to a Timed Automata model and validated in the modeling tool UPPAAL [25]. A formal mechanism to convert Time Petri Nets to corresponding Timed Automata models has been given by Gu and Shin [11]. However, all these prior works on Time Petri Nets deal only with the preemptive or non-preemptive models of execution and cannot be directly applied for schedulability analysis in P-FRP.

9. Conclusions

P-FRP is a new functional programming formalism for implementing real-time embedded systems. Due to the state-less execution of functional programs, preempted tasks in P-FRP have to be aborted. The abort of preempted tasks leads to an execution semantics which is different from preemptive or non-preemptive execution and makes the schedulability analysis of this model complicated. We have presented TPN models for the schedulability analysis of P-FRP and shown that TPN offers an efficient alternative to existing polynomial time methods.

Even though the transactional execution model is more complex than the classical preemptive model, we feel that use of TPN's for the schedulability analysis of P-FRP is actually much easier that using TPN's in the preemptive model. This is because in the latter, the number of execution steps performed by preempted tasks has to be stored and several extra transitions and places have to be added for this purpose. Since preempted tasks have to restart there is no need to 'remember' the execution steps in P-FRP.

In future work, we will be modifying our TPN models to do schedulability analysis lock-free execution [1] and transactional memory [13], which are execution models that use similar transactional execution semantics. Analysis of efforts involved to scale up the model for multi-processor system will also be undertaken.

References

[1] J. H. Anderson, S. Ramamurthy, K. Jeffay. "Real-time computing with Lock-free Shared Objects". *ACM Transactions on Comp.Sys. 5(6), pp.388-395*, 1997.

[2] C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP". *Practical Aspects of Declarative Languages (PADL)'11, Lecture Notes in Computer Science, Springer*, 2011.

[3] C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP using Idle-Period Game Board". *IEEE ISORC'11*, 2011.

[4] C. Belwal, A.M.K. Cheng. "Schedulability Analysis of P-FRP using Time Petri Nets". *RTCSA'11 WiP session,* 2011.

[5] C. Belwal and A.M.K. Cheng. "An Extensible Framework for Real-time Task Generation and Simulation". *IEEE RTCSA'11*, 2011.

[6] H. Boucheneb,G. Garde, O. H. Roux. "TCTL Model Checking of Time Petri Nets". *Journal of Logic and Computation, vol. 90, pp. 1509-1540*, 2009.

[7] C. Elliott, P. Hudak. "Functional reactive animation". *ICFP'97, pp. 263-273,* 1997.

[8] E.A. Emerson. "Temporal and modal logic". *Handbook of Theoretical Computer Science, Jan van Leeuwen, vol. B. MIT Press. pp. 955–1072*, 1990.

[9] A. Furfaro, L. Nigro. "Modelling and Schedulability Analysis of Real-time Sequence Patterns using Time Petri Nets and UPPAAL". *International Workshop on Real Time Software (RTS'07)*, *October 16 , pp. 821-835*, 2007.

[10] G. Gardey, D. Lime, M. Magnin, O.H. Roux. "Roméo: A tool for analyzing time Petri nets". *CAV'05, Lecture Notes in Computer Science, Springer*, 2005.

[11] Z.Gu, K.G. Shin. "Analysis of Event-Driven Real-Time Systems with Time Petri Nets: A Translation-Based Approach". *Design and Analysis of Distributed Embedded Systems DIPES '02*, *pp.31-40*, 2002.

[12] Haskell, *http://www.haskell.org.*

[13] M. Herlihy, J.E.B. Moss. "Transactional memory: architectural support for lock-free data structures". *ACM SIGARCH Computer Architecture New (Col. 21, Issue 2), pp.289-300,* 1993.

[14] R. Kaiabachev, W. Taha, A. Zhu. "E-FRP with Priorities". *EMSOFT'07*, *pp.221-230*, 2007.

[15] C. L. Liu, L. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM (Volume 20 Issue 1)*, *pp. 46-61*, 1973.

[16] P.M. Merlin, D.J. Farber. "Recoverability of Communication Protocols - Implications of a Theoretical Study". *IEEE Transactions on Communications, vol.24, pp.1036-1043*, 1976.

[17] T. Murata**. "**Petri nets: Properties, analysis and applications", *Proceedings of the IEEE, Vol. 77 Issue 4,pp. 541-580*, 1989.

[18] J. Peterson, G. D. Hager, P. Hudak. "A Language for Declarative Robotic Programming". *IEEE ICRA'99*, 1999.

[19] J. Peterson, P.Hudak, A.Reid, G. D. Hager. "FVision: A Declarative Language for Visual Tracking". *PADL'01,* 2001.

[20] C. Ramchandani, "Analysis of asynchronous concurrent systems by Timed Petri Nets," *MIT, Project MAC, Tech. Rep. 120*, 1974.

[21] J. Ras, A.M.K. Cheng. "Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm". *RTCSA'09*, 2009.

[22] J.Ras, A.M.K. Cheng. "Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing". *ICESS-'10*, 2010.

[23] L. Sha, R. Rajkumar, J. P. Lehoczky. "Priority Inheritance Protocols: An approach to Real Time Synchronization". *Transactions on Computers Volume 39, Issue 9, pp.1175-1185,* 1990.

[24] J.J. P. Tsai, S. J. Yang, Y.H. Chang. "Timing Constraint Petri Nets and Their Application to Schedulability Analysis of Real-Time System Specifications". *IEEE Trans. Softw. Eng. 21, 1 (January 1995)*, *pp. 32-49*, 1995.

[25] *http://www.uppaal.org/*

[26] Z. Wan, W. Taha, and P. Hudak. "Real - time FRP". *ICFP'01*, *pp. 146-156*, *ACM Press*, 2001.

[27] Z. Wan, W. Taha, P. Hudak. "Task driven FRP". *PADL'02*, 2002.

[28] Z. Wan, P. Hudak. "Functional reactive programming from first principles". *Programming Language Design and Implementation*, *pp.242-252*, 2000.

[29] D.Xu, X.He,Y.Deng. “Compositional schedulability analysis of real-time systems using time Petri nets”. *IEEE Trans. On Software Engineering, vol 28, issue 10, pp. 984-996*, 2002.