Using Timed Petri Nets to Model Spatial-temporal Group Scheduling Problems

Daniel Graff[1], Tammo M. Stupp[1], Jan Richling[1], and Matthias Werner[2]

[1] Communication and Operating Systems Group,
Berlin Institute of Technology, Germany
{dgraff,stupp,richling}@cs.tu-berlin.de
[2] Operating Systems Group, Chemnitz University of Technology, Germany
matthias.werner@informatik.tu-chemnitz.de

Abstract. Dealing with cyber-physical systems (CPS) puts a strong emphasis on the interaction between computing and non-computing elements. Since the physical world is characterized by being strongly distributed and concurrent, this is also reflected in the computational world making the design of such systems a challenging task. If a number of tasks shall be executed on a CPS where each task is bound to time and space, may have dependencies to other tasks and requires a specific amount of computing devices, a solution requires a four-dimensional space-time schedule which includes positioning of the devices resulting in an NP-hard problem.
In this paper, we address the problem of spatial-temporal group scheduling using Timed Petri nets. We use Timed Petri nets in order to model the spatial, temporal, ordered and concurrent character of our mobile, distributed system. Our model is based on a discrete topology in which devices can change their location by moving from cell to cell. Using the timed property of Petri nets, we model movement in a heterogeneous terrain as well as task execution or access to other resources of the devices. Given the modeling, we show how to find an optimal schedule by translating the problem into a shortest path problem, which is solvable with the known method of dynamic programming.

1 Introduction

Analyzing the technical evolution starting from over half a century ago up to nowadays, it is obvious that computational devices become more powerful (according to Moore's law), get smaller in size and as a rather recent trend mobility and pervasive computing gain a major issue in the everyday life of people. By becoming increasingly interconnected and communicative, these devices form new types of intelligent distributed systems that are aware of themselves, their surroundings (using sensors), and their capabilities to manipulate the former two (using actuators). In particular, a strong interaction with the physical world as performed, e.g., by mobile robots, involved in the same global system lead to the emergence of cyber-physical systems (CPS) where "*physical processes affect* *computations and vice versa*" [5]. By focusing on the physical world it becomes obvious that non-computational processes (physical actions) are strongly distributed and concurrent. Thus, designing and programming those systems have to cope with those issues. Since thinking in distributed and concurrent terms is complexity-introducing and often error-prone [6], we have studied this problem and proposed a suitable programming model [2, 3, 4] that both abstracts from distribution and concurrency by allowing the programmer to develop sequential object-oriented program code. Besides the imperative code fragments, declarative annotations can be integrated into the source code for defining spatial-temporal constraints that are glued to imperative code fragments and restrict its execution.

In our approach, we propose a systemic view in order to describe the impact on non-computing (physical) entities of the real world that in classical embedded system design is called "controlled object". In our programming model, we represent those non-computing entities by objects[3]. Based on the concept of object-oriented program design, a method is responsible for interacting with the object. Therefore, we use methods that we call *actions* that represent the interface between computing and non-computing elements. In particular, actions invoked in the cyber world result in physical actions by means of sensors/ actuators that affects the controlled object. Actions may involve multiple sensors/ actuators that have to be used at a specific time at a specific physical location. Generally speaking, this requires a coordination of resources in space and time. Therefore, in this paper, we address the problem of spatial-temporal group scheduling by using Timed Petri nets. Our concept is to map space to time and describe physical locations based on durations needed to change locations. The computation of a schedule is based on those timed transitions.

The remainder of the paper is structured as follows: In Section 2, we provide a basic understanding of our assumptions about the domain we are targeting including a problem statement and the overall goal. In order to formalize our problem and build up a model, Section 3 introduces Timed Petri nets as a well-known method to model dynamic systems with discrete states followed by Section 4 which shows the modeling elements. Section 5 provides an analysis of the modeled problem by finding an optimal schedule by mapping to the shortest path problem. Finally, Section 6 concludes the paper and gives an overview about future work.

2 Assumptions

In our understanding, a task is associated with a duration and, e.g., a deadline at which the task has to be completed depending on hard or soft deadlines. Thus, tasks may have temporal constraints. In this paper, we extend the classical view by a new dimension: space.

[3] In terms of a programming language

A task t is described by a set of properties $\{d, p, p', r, T'\}$, with d indicating the duration[4] of the task and p and p' the beginning and ending location of the task, respectively. A task may also bound to a fixed location—in that case p and p' are identical. A location is a physical position on a 2D surface. In addition, we address the problem of performing tasks jointly, i.e., a given amount of robots $r \in R$, with $|R|$ denote the total number of robots, is required to perform a task that have to be coordinated in space and time. For simplification, we assume tasks are non-interruptable. Finally, the execution of t depends on the result of the set of predecessor tasks T' that need to be executed prior to t.

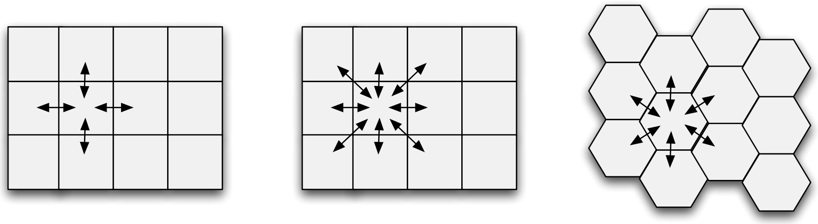
Fig. 1. Examples for discrete topologies

The 2D surface in which the robots operate is discretized and mapped to a specific topology. Each cell c_i in the topology indicate a space in which an arbitrary amount of robots can be placed: $c_i \in \{x \in \mathbb{N} \mid 0 \leq x \leq |R|\}$ and $\sum c_i = |R|$. We support different topologies as shown in Fig. 1 with respect to the geometry of the surface, the discretization (cell shape) and the multiplicity of movements. A robot can change its location by moving in discrete steps to a neighboring cell along the indicated arrows. On the left hand side of the figure a cell is represented by a square and exhibits four possible movements of a robot. The middle topology doubles the degree of freedom by allowing diagonal movements. Finally, the topology on the right shows a discretization that is based on hexagons which allows for six different types of movements. During each time step a robot has different options:

- Stay in the current cell (idle)
- Move along the arrows towards a neighboring cell
- Execute a task (if the task involves movement, the robot moves towards the tasks' ending location while executing it at the same time)

The movement model is based on a binary state: The robot does not move (idle) or simply moves (speed is not incorporated in the model). If a robot decides to move to a neighboring cell, the cell transition is associated with a given amount of time required for reaching the other cell (this again represents

[4] In this paper we always assume the worst case duration, i.e., the worst case execution time.

the worst case time needed for moving the robot). The topology does not have a homogeneous terrain, thus, times between cell transitions may vary. With this, we are able to model accessible and non-accessible obstacles. Driving uphill takes more time to process the transition than driving downhill. On the other hand, a solid formation, e.g. rocks, are not accessible and, thus, the robots have to take the longer way in terms of geographic distance. Altogether, this approach allows us to model the important properties of robots moving in a terrain without the need to deal with the physics of the actual movement actions—these are represented by the time needed for transitions between the cells.

Now, the overall goal is to find a schedule with minimal makespan such that all tasks t_i are executed according to their requirements of beginning and ending location and the number of robots which includes physical positioning of robots.

3 Timed Petri Nets

Petri nets [8] are a well-known method to model dynamic systems with discrete states. The original Petri net does not include a notion of time. However, there exist a number of extensions to Petri nets enabling them to deal with time. The probably most two famous approaches are *Time Petri nets* [7] and *Timed Petri nets* [9], also called *Duration Petri nets* (DPN). In this paper, we use the later one[5]: A Timed Petri net or Duration Petri net is a structure $N = (P, T, F, V, D, m_0)$, where

- P, T, F are finite sets with $P \cap T = \emptyset$, $P \cup T \neq \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ and $dom(F) \cup cod(F) = P \cup T$, where the elements $p \in P$ are called *places*, the elements $\tau \in T$ are called *transitions*, and the elements of F are called *arcs*.
- $V : F \to \mathbb{N}$ is a weight of the arcs
- $D : T \to \mathbb{N}$ is a *duration function* so that $D(\tau)$ denotes the delay of the transition τ.[6]
- A mapping $P \to \mathbb{N}$ is called a *marking* of the net. We say the marking assigns to each place a number of *tokens*. m_0 is the *initial marking*.

The marking of a TPN may change considering the following rules:

- A transition τ is *enabled* at marking m iff $\forall p \in P, (p, \tau) \in F, m(p) \geq V(p, \tau)$.
- If at least one enabled transition exists, transitions of the TPN must *fire*.[7]
- During firing of a transition τ, tokens are removed and added:
  - At firing time t_0: $\forall p \in P, (p, \tau) \in F, m'(p) = m(p) - V(p, \tau)$
  - At time $t_0 + D(\tau)$: $\forall p \in P, (\tau, p) \in F, m'(p) = m(p) + V(\tau, p)$

[5] Please note that differing from the definition given in [9], we allow zero time duration and a transition may fire immediately again

[6] Usually, a mapping $T \to \mathbb{Q}$ is considered; however, it is easy to see, that considering TPN with $D : T \to \mathbb{N}$ will not result in a loss of generality.

[7] Please note the difference to original Petri nets: there, an enabled transition *may* fire.

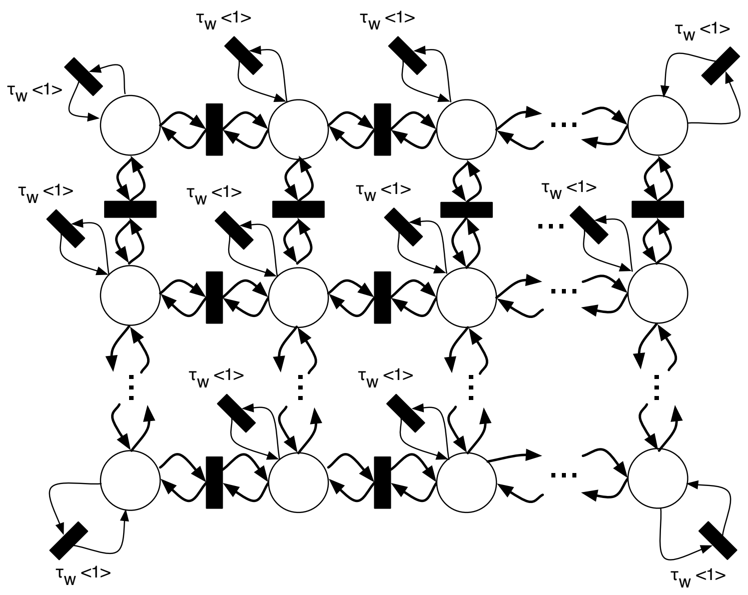

Fig. 2. Model of a grid topology (all transitions are timed)

Please note, that our definition abandons transition clocks, but considers the transition delay by the firing semantic. This allows self-concurrency for firing of a transition. I.e., a transition that stays enabled after initiate a firing may *immediately* (i.e., without elapsing of time) start firing again.

4 Modeling

In this section, we describe how to set up a model for our problem of spatial-temporal group scheduling problems based on the assumptions given in Section 2 by means of Timed Petri nets as presented in Section 3. We split our model into two parts: Physical movement in a 2D area and constrained task execution before we put both together and glue it into one model.

Let's start to find a suitable abstraction in order to describe physical locations in which robots operate. As described in Section 2, the 2D surface is discretized and mapped to a specific topology (cf. Fig 1). Arrows indicate possible movements in order to reach neighboring cells. We model this topology by means of Timed Petri nets. Each cell of the topology is mapped to a single place[8] of the net and each arrow is mapped to a transition[8] as shown in Fig. 2 for the grid. As common for a TPN, transitions are timed (denoted in sharp brackets). By using timed transitions, we model how much time is required by a robot in order to change its current location Our concept is to map space to

[8] Places and transitions in terms of Petri nets

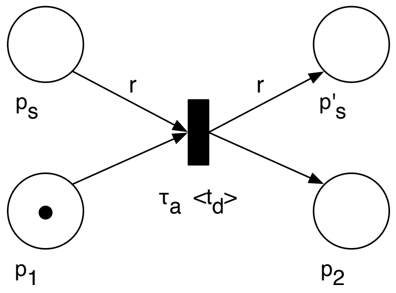

Fig. 3. Modeling an activity

time and describe physical locations based on durations needed to change locations. However, tokens represent the robots and, thus, also their current physical location.

Based on the characteristic of a TPN—an enabled transition must fire—the robots would be forced to move. Based on our assumptions of Section 2 a robot may also stay calm at its current location for an arbitrary amount of time. Therefore, we model an additional transition τ_w that allows to model such behavior.

We define an activity as a task that shall be executed by one or several robots and takes a certain amount of time for execution. We model this by using a timed transition τ_a with a duration of t_d time units as shown in Fig. 3.

Places p_1 and p_2 reflect the status of the activity, i.e., a token on p_1 marks the activity as "ready to run" while p_2 states that the activity has been executed. With this modeling an activity can only be executed once. Furthermore, an activity is bound to a physical location at which it shall be executed and requires a given amount robots that need to be incorporated in the common execution. Place p_s[9] denotes the physical location at which the execution shall be started and place p'_s denotes the location at which the execution shall end. If $p_s = p'_s$ then the robots do not move while executing the activity. The parameter r on the arc between p_s, τ_a and τ_a, p'_s denotes the amount of robots incorporated in the activity. Modeling dependencies between two activities A_1 and A_2 can be easily done by using p_2 of A_1 as p_1 of A_2. Enabling τ_a of A_2 can, therefore, only be done if p_2 of A_1 is marked.

Now, we have to put both models together resulting in one TPN as shown in Fig. 4—due to the construction of our model elements, this composition is straightforward. There, we have in total four robots positioned on the physical grid. Two of the robots stay at the same physical location. Assuming $r = 4$ and τ_a shall be enabled, requires the other two robots to move to the upper left place. If transition τ_a fires, place p_2 is marked and r tokens are put on the place in the middle of the grid. Assuming, we have multiple activities, the overall goal is to

[9] For clarity places denoted with index s stands for physical locations (space)

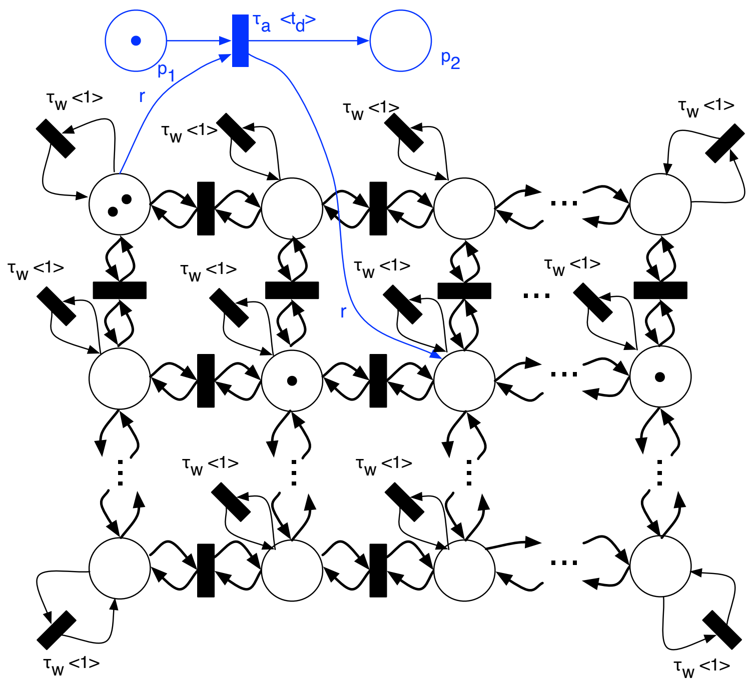

Fig. 4. Modeling a complete scheduling problem

mark every p_2 of each activity with a token indicating that we have executed all activities.

5 Analysis

After modeling the problem, we can find an optimal schedule by translating the problem into a shortest path problem, which is solvable with the known methods of dynamic programming, cf., e.g., [1, 11].

For this translation, we use a similar approach we have chosen in [10]. There, the optimization parameter was the reward, mapped to number of tokens at certain places, and time (deadline) was a side condition. Here, time is the optimization parameter and a certain marking is the side condition.

The straightforward solution is to spawn the state graph with transition delays as arc weight, seek the state(s) that has the places p_2 marked for all activities (goal states), and use dynamic programming to find the shortest path to a goal state. This path represents the optimal schedule. However, the state space might be infinite, even if the Petri net itself is finite. Also, it might be possible that no goal state is reachable.

If we consider the structure of a model net instance, we can make the observation that all transitions are conservative. I.e., firing never change the overall number of tokens. Because of this observation, the overall net is conservative, too, $\forall m, \mathit{reachable}(m), |m| = |m_0|$[10], and following, the (timeless) state space of the net is finite. Thus, there is no need to add another timed edge (s_1, s_2) to the state space graph, if there is already an edge between s_1 and s_2, i.e., the first edge covers all other edges regarding a shortest path algorithm. Following, the "interesting" state graph is finite.

Let $r_{max} = \max(r_i)$ and n_a the number of activities. Obviously, no goal state can reached if $|m_0| < n_a + r_{max}$. In this case, there exist no feasible schedule. If there are enough (i.e. at least r) tokens at location places, it is easy to see that a goal state is always reachable.[11]

The complexity of spawning the state space is exponential, whereas finding the schedule has a polynomial complexity in the number of states in the state space. However, the shortest path algorithm can be done during the spawning, c.f. [10]. Then, the first found goal state terminates the spawning.

6 Conclusion

Designing CPS is a challenging task since it has to consider interactions between several computing and non-computing elements, and, thus, consideration of concurrent and distributed tasks. In particular, if a task has to be executed jointly by multiple robots and in addition is bound to a specific physical location, this requires a coordination of robots in space and time. Furthermore, considering multiple tasks, with individual demands in terms of execution time, number of robots, dependencies between tasks and begin and end location requires a suitable spatial-temporal group scheduling.

In order to deal with these problems, an appropriate modeling technique is needed. In this paper, we propose to use Timed Petri nets to model the spatial-temporal properties of such systems. The model itself is based on a discrete topology where each cell represents a physical location on a 2D surface. This topology is represented by a Timed Petri net where the transitions between the cells represent the movement of robots in a simplified way by assigning times to these transitions. Similarly, local executions are modeled based on the time they need. This model allows to map our problem of finding a spatial-temporal schedule for a CPS to the problem of finding a path such that all given tasks have been executed and the time is minimized—a shortest path problem. Therefore, we are able to apply known methods of dynamic programming to solve the problem.

As future work, we plan to consider the following directions: We want to incorporate different types of robots by using Coloured Petri nets enabling to model robots with different properties (e.g., different speed, different capabilities with respect to terrain, different capabilities to execute applications, etc.). Furthermore, similar to common operating system tasks, we want to support

[10] For a given marking m, $|m|$ denotes the number of tokens

[11] Obviously in the timeless net; in the timed net because of τ_w as loop at all location places.

suspending and resuming the execution of a task. Finally, we aim to define and describe a general approach to space-time scheduling which is be based on a continuous space and may include deadlines and, e.g., specific formations of robots.

References

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, 2nd edn. (2001)
[2] Graff, D., Richling, J., Stupp, T.M., Werner, M.: Context-aware annotations for distributed mobile applications. In: Wolfgang Karl, D.S. (ed.) ARCS'11 Workshop Proceedings: Second Workshop on Context-Systems Design, Evaluation and Optimisation (CoSDEO 2011). pp. 357–366. VDE (February 2011)
[3] Graff, D., Richling, J., Stupp, T.M., Werner, M.: Distributed active objects – a systemic approach to distributed mobile applications. In: Sterrit, R. (ed.) 8th IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems. pp. 10–19. IEEE Computer Society (April 2011)
[4] Graff, D., Werner, M., Parzyjegla, H., Richling, J., Mühl, G.: An object-oriented and context-aware approach for distributed mobile applications. In: Workshop on Context-Systems Design, Evaluation and Optimisation (CoSDEO 2010) at ARCS 2010 - Architecture of Computing Systems. pp. 191–200 (2010)
[5] Lee, E.A.: Cyber-physical systems – are computing foundations adequate? In: Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap (October 2006)
[6] Lee, E.A.: The problem with threads. Computer 39, 33–42 (2006)
[7] Merlin, P.: A Study of the Recoverability of Communikation Protocols. Ph.D. thesis, University of California, Computer Science Department, Irvine, CA, USA (1974)
[8] Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Universität Bonn, Institut für Instrumentelle Mathematik, Bonn (1962)
[9] Ramchandani, C.: Analysis of asynchronous concurrent systems by Timed Petri Nets. Project MAC-TR 120, MIT, Massachusetts Institute of Technology, Cambridge, MA, USA (Feb 1974)
[10] Werner, M.: A framework to find optimal iris schedules. Journal of Control and Cybernetics 35(3), 703–719 (2006)
[11] Yee, S.T., Ventura, J.A.: A dynamic programming algorithm to determine optimal assembly sequences using petri nets. International Journal of Industrial Engineering - Theory, Applications and Practice 6(1), 27–37 (1999)