Modeling Group Scheduling Problems in Space and Time by Timed Petri Nets

Daniel Graff, Jan Richling
Communication and Operating Systems Group
Berlin Institute of Technology, Germany
dgraff@cs.tu-berlin.de
richling@cs.tu-berlin.de

Matthias Werner
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 which are 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 time 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.

Keywords: Spatial-temporal group scheduling problems, scheduling, Timed Petri nets, cyber-physical systems

Address for correspondence: Technische Universität Berlin, Sekretariat EN 6, Einsteinufer 17, 10587 Berlin
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” [12]. 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 [13], we have studied this problem and proposed a suitable programming model [9, 10] that both abstracts from distribution and concurrency and allows the programmer a systemic view.

Our approach sketched above requires a coordination of resources (robots) in space and time. Therefore, in this paper, we address the problem of spatial-temporal group scheduling. A key factor for the modeling is concurrency since robots may perform operations in parallel at the same point in time jointly or fully independent. Thus, we chose Petri nets [17] since it is a well-known method to model dynamic systems with discrete states and it is well suited for modeling the concurrent behavior of distributed systems.

We model terrain and task properties and, finally, glue these two models together resulting in one Petri net. After modeling the problem, the overall goal is to find an optimal space-time schedule, i.e., a schedule for the given tasks with the time as optimization parameter. For this, we translate the problem into a shortest path problem. The shortest path represents the time optimal schedule.

The remainder of the paper is structured as follows: We start with considering related work in Section 2. In Section 3, 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 4 introduces Timed Petri nets followed by Section 5 which describes our model in detail. Section 6 provides an analysis of the modeled problem by finding an optimal schedule by mapping to the shortest path problem. Finally, Section 8 concludes the paper and gives an overview about future work.

2. Related Work

Designing CPS is a challenging task and raises a number of issues [12]. There are numerous implementations of context-aware middleware systems supporting mobility, such as GaiaOS [22], SuperSpaces [2], QoSDream, to name a few. However, even if several of them facilitate GPS data to determine locations, none of the approaches support real-space distribution, real space-time scheduling or a system view for distributed applications. An approach that has similar objectives like ours is SwarmOS [14]. It supports groups of nodes (especially robots) that collectively execute applications without the need to program the nodes separately. Since SwarmOS is intended to support swarm algorithms with stochastic elements, it may not be suitable to fit our needs as we require more deterministic algorithms.
In general, there are numerous of Petri net publications. 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 [15] and Timed Petri nets [20] (TPN), also called Duration Petri nets (DPN). Besides discrete Petri nets, e.g., TPN and DPN, there are also Continuous Petri nets [5] and Hybrid Petri nets [3] which are typically used in order to model a system which has a discrete part (e.g., Boolean state variable) and a continuous part (e.g., real number indicating liquid flow). A good survey about Hybrid Petri nets including Timed Hybrid Petri nets is presented in [6]. A good overview about Petri nets including news and tools is provided from the TGI group from the University of Hamburg [23]. Many applications of Petri nets target scheduling problems, e.g., deadlock avoidance, performance evaluation, or feasibility of real-time and non-real-time problems or workflow scheduling [25]. There is also work that uses dynamic programming [26] to solve an assembly line scheduling problem described by Petri nets as well as a shortest/longest path problem used to find optimal real-time schedules [18]. However, none of the publications in the Petri net literature deals with spatial-temporal group scheduling which is a fundamental necessity for our project.

3. Assumptions and Model

Usually, a task on a computer system is associated with a duration, i.e., the time the task needs for being executed. In case of real-time systems, this simple model is extended: The worst case duration of the task is considered and there is a deadline at which the task has to be completed with soft or hard requirements as needed by the application. Sometimes, additional restrictions regarding the start time of the task apply. Thus, tasks may have temporal constraints. In this paper, we extend this view by a new dimension: space.

Conceptually, this means that our tasks now have space-time requirements for both start and completion as well as an upper bound on execution time. Execution units are able to move in space while executing tasks—we assume them to be robots and declare them as mobile resources. Although our approach is targeted towards this model, for simplicity reasons we consider a simpler model in this paper: Execution is only restricted by location requirements, space is considered to be a discrete 2D surface and time is only involved as worst case durations. In Section 8 we will briefly sketch how to remove these restrictions.

Formally, a task $\gamma$ is now described by a set of properties $\{d, p, p', r, \Gamma^\prime\}$, with $d$ indicating the duration of the task and $p$ and $p'$ the beginning and ending locations 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 in real space. In addition, we address the problem of performing tasks jointly, i.e., a given amount of robots $r \leq |\Upsilon|$, with $\Upsilon$ being the set 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 $\gamma$ depends on the result of the set of predecessor tasks $\Gamma^\prime$ that need to be executed prior to $\gamma$.

The 2D surface in which the robots operate is assumed to be discretized and mapped to a specific topology. Each cell $c \in C$ in the topology indicates a space in which an arbitrary amount of robots can be placed. The amount $a$ of robots per cell can change over time: $a(c, t) \in \{x \in \mathbb{N} \mid 0 \leq x \leq |\Upsilon|\}$

---

1. Although we eliminate the third dimension of movements by doing so, we do not require the space to be 2D only: The 2D surface is embedded in 3D space allowing locations to have different altitudes. We only restrict our robots to move on ground, not to fly or dig.

2. In this paper we always assume the worst case duration.
and for each $t$ holds: $\sum_{c \in C} a(c, t) = |\Upsilon|$, with $|\Upsilon|$ equals the number of available robots. 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. In analogy to our discrete space model we also use a simplified discrete time model: Time progresses in time steps. We argue for a discrete space and time model for two reasons: First, we are able to control the complexity based on the granularity of the discretization (since we are dealing with resource restricted embedded systems, we have to reduce complexity). Second, we have an inaccuracy concerning robot movement. Thus, we have an upper bound for the resolution. Exceeding a certain threshold by a more fine-grained discretization does globally not lead to better test results. The same applies to a continuous space and time model.

The movement model is based on a binary state: A robot moves or does not move. We assume that our topology does not have a homogeneous terrain so movements from one cell to another may need different times depending on terrain. With this, we are able to model accessible and non-accessible obstacles. Driving uphill takes more time than driving downhill. On the other hand, a solid formation, e.g. rocks, is 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 moving between the cells.

Therefore, at each time step a robot that is not currently involved in a task execution (started at an earlier time step but not yet finished) has the following options: The robot may stay calm in its current cell (idle), move along the arrows towards a neighboring cell, execute a task while not moving or execute a task while moving along the arrows towards a neighboring cell (in this case, the robot moves towards the tasks’ ending location while executing it at the same time).

Overall, we have a set $\Gamma$ of tasks. The goal is to find a real-space-time schedule with minimal makespan such that these tasks $\gamma_i \in \Gamma$ are executed according to their requirements.

4. Timed Petri Nets

In general, time is a necessity for scheduling and since we deal with space-time scheduling, we chose TPN since they include time and they are a strict technique which is most suitable for our approach. The formalism used for the remainder of the paper has been adapted from [19].

We use the following notation: $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ is the set of natural numbers $\mathbb{N}$ without 0; $\mathbb{Q}$ is the set of rational numbers; $\text{dom}(f)$ and $\text{codom}(f)$ is the domain resp. codomain of a function $f$. $M(m, n)$ is
a matrix with \( m \) rows and \( n \) columns. For a matrix \( M \), \( M^{(i)} \) denotes the \( i \)-th row of \( M \), starting with 0. \( M^{(j)} \) denotes the \( j \)-th column of \( M \), starting with 0 and \( M^{(i,j)} \) denotes the \( i \)-th row and \( j \)-th column of \( M \).

**Definition 1. (Timed Petri net (TPN))**

A TPN or DPN is a graph \( N = (P,T,F,V,D) \), 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 \( \text{dom}(F) \cup \text{cod}(F) = P \cup T \), 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*.
- \( V : F \rightarrow \mathbb{N}^+ \) is a weight of the arcs.
- \( D : T \rightarrow \mathbb{N}^+ \) is a *duration function*. \( D(\tau) \) denotes the delay of transition \( \tau \).

**Definition 2. (State of a TPN)**

The state of a TPN is a tuple \( S = (m,h) \), where

- \( m : P \rightarrow \mathbb{N} \) is called a *marking* of the net. We say the marking assigns to each place \( p \) a number of *tokens* denoted by \( m(p) \).
- \( h : T \rightarrow \mathbb{N} \) is a transition clock vector. \( h(\tau) \) denotes the clock vector of the transition \( \tau \).

A TPN has a dedicated state \((m_0,h_0)\), called *initial state*. In an initial state, \( h_0 \) is always the zero vector. When the dynamics of a TPN is considered (see below), the initial state serves as a starting point of all considerations.

To any transition \( \tau \in T \) belongs a pre-set \( \bullet \tau \) and a post-set \( \tau \bullet \), that are given as \( \bullet \tau = \{ p \mid p \in P \land (p, \tau) \in F \} \) and \( \tau \bullet = \{ p \mid p \in P \land (\tau, p) \in F \} \), respectively. Each transition \( \tau \in T \) induces the markings \( \tau^- \) and \( \tau^+ \), defined as follows:

\[
\tau^- = (v_1, \ldots, v_{|P|})^T \mid v_i = \begin{cases} V(p_i, \tau) & \text{if } (p_i, \tau) \in F \\ 0 & \text{else} \end{cases}
\]

\[
\tau^+ = (v_1, \ldots, v_{|P|})^T \mid v_i = \begin{cases} V(\tau, p_i) & \text{if } (\tau, p_i) \in F \\ 0 & \text{else} \end{cases}
\]

Then \( \Delta \tau \) denotes the difference \( \tau^+ - \tau^- \). A transition \( \tau \) is *enabled* at marking \( m \) iff \( \tau^- \leq m \) and \( h(\tau) = 0 \).

In order to introduce a state equation later, we extend the definition of a marking to a *time marking*:

**Definition 3. (Time Marking)**

A time marking \( \mu \in M(|P|,d) \) is a matrix; with \( d = \max\{D(\tau_i)\mid i \in \{1, \ldots, |T|\} \} + 1 \), where each row represents a specific place \( p \). Each column denotes a (partial) marking of a place for different time steps. The first column represents the present (at time \( t \)), i.e., equals the actual marking. The second one denotes partial changes (i.e., future additions from time elapsing) in the net at \( t + 1 \) (not the marking at \( t + 1 \)). The third column then represents partial changes at \( t + 2 \) and so on.

Please note, that there exist a mapping \( S \rightarrow \mu \) and a mapping \( \mu \rightarrow m \). However, no mapping \( \mu \rightarrow S \) exists.

\(^3\text{Usually, a mapping } T \rightarrow \mathbb{Q} \text{ is considered; however, it is easy to see, that considering TPN with } D : T \rightarrow \mathbb{N}^+ \text{ will not result in a loss of generality.} \)
Given an example net as shown in Fig. 2 and assumed the net is in an initial state, the time marking

\[ \mu_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \].

Now we are ready to consider TPN’s dynamics. A TPN can change its state by firing and by time elapsing. We denote elements that are changed by firing with a hat (e.g., \( \hat{m} \)) and elements that are changed by time elapsing by a tilde (e.g., \( \tilde{m} \)). Since we will use time marking in our analysis (c.f. Section 6) and for the sake of clarity, we consider the impact to the time marking together with the change of the state. Since a TPN is enforced to progress, we introduce the maximal step.

**Definition 4. (Maximal Step)**

\( \mathcal{B} \subseteq \mathcal{T} \) is called a maximal step at the marking \( m \) (resp. at time marking \( \mu \), since \( m = \mu^{(0)} \)) with the transition clock vector \( h \) iff

\[ \sum_{\tau \in \mathcal{B}} \tau^- \leq m \]  \( (1) \)

\[ \forall \tau, \tau' \in \mathcal{B} \rightarrow h(\tau) = 0 \]  \( (2) \)

\[ \neg \exists \mathcal{B}' ((\mathcal{B} \subseteq \mathcal{B}') \land (\mathcal{B}' \text{ satisfies } (1) \text{ and } (2))) \]  \( (3) \)

If at least one enabled transition exists, transitions of the TPN must fire. Only maximal steps fire in a TPN. If there are more than one maximal step that may fire, one of them is selected arbitrarily. The example net shown in Fig. 2 with the initial marking \( m_0 \) has four maximal steps:
\[ B_1 = \{ \tau_1, \tau_3 \} \]
\[ B_2 = \{ \tau_1, \tau_8 \} \]
\[ B_3 = \{ \tau_4, \tau_3 \} \]
\[ B_4 = \{ \tau_4, \tau_8 \} \]

**Definition 5. (Firing)**
A TPN with the time marking \( \mu \) and with a maximal step \( B \) that becomes enabled at time \( t \) will change its state in the following way.

\[
\hat{m} = m - \sum_{\tau \in B} \tau^-
\]

\[
\forall \tau \in B, \hat{h}(\tau) = D(\tau)
\]

\[
\forall k \in \{1, \ldots, d\}, \hat{\mu}^{(k)} = \mu^{(k)} + \begin{cases} - \sum_{\tau \in B} \tau^- & \text{iff } k = 0 \\ \sum_{\tau \in B : D(\tau) = k} \tau^+ & \text{else} \end{cases}
\]

The example net (Fig. 2) has four possible maximal steps on \( \mu_0 \). Let \( \mu_1 \) be the time marking which the net reaches after firing \( B_2 \), i.e., \( \mu_0 \xrightarrow{B_2} \hat{\mu}_1 \). It is easy to see that

\[
\hat{\mu}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \hat{m}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{h}_1 = \begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\]

\( B_2 \) indicates firing of \( \tau_1, \tau_8 \) which involves removing tokens from \( p_1 \) and \( p_4 \) at time \( t \). Thus, the first column of \( \hat{\mu} \) contains only zeros (tokens “reside” in the transitions). The duration of \( \tau_8 \) is one time unit. Thus, at \( t + 1 \) (second column) one token is released and put on \( p_3 \). The duration of \( \tau_1 \) is four time units (longest firing duration of all \( T \)) and, therefore, at \( t + 4 \) (fifth column) the remaining token appears on \( p_2 \).

For the elapsing of time, we use a specific matrix called *progress matrix* \( Q \). The progress matrix \( Q \in M(d, d) \) is of the form

\[
Q = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{pmatrix}
\]
Definition 6. (Time Elapsing)
Given a TPN with a marking \( \hat{m} \), a clock vector \( \hat{h} \), and a time marking \( \hat{\mu} \). When one time unit elapses, the values change in the following way:

\[
\tilde{m} = \hat{m} + \sum_{\tau \in T | \hat{h}(\tau) = 1} \tau^+
\]

(7)

\[
\tilde{h} = \max(\hat{h} - 1, 0)
\]

(8)

\[
\tilde{\mu} = \hat{\mu} \cdot Q
\]

(9)

Applied to our example, we get:

\[
\tilde{\mu}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \tilde{m}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad \tilde{h}_1 = \begin{pmatrix} 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{and} \quad \tilde{h}_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.
\]

5. Translating Model into TPN

In this section, we describe how to model our problem of spatial-temporal group scheduling problems based on the assumptions given in Section 3 by means of Timed Petri nets. We split our model into the parts “physical movement in a 2D area” and “constrained task execution” before we put both together.

Let us start to find a suitable abstraction in order to describe physical locations in which robots operate. As described in Section 3, 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 of the net and each arrow is mapped to a transition as shown in Fig. 3(b) 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 in the worst case. Considering Fig. 2 from Section 4 again, we map this simple net onto a surface as depicted in Fig. 3(a). By doing so, we are able to model an area where two places \((p_2, p_4)\) have a higher elevation than the other two \((p_1, p_3)\). Therefore, it takes significantly more time to move uphill while going downhill is faster – this is reflected by the transition durations. The locations of the robots are therefore mapped to markings of the places that model the individual cells, with other words, the robots itself are represented by the tokens on these places.

Beside moving, a robot may also stay idle at its current location for an arbitrary amount of time. Since the firing strategy of our TPN is maximal step a robot would be forced to move. In order to avoid continuous movement, we introduce an additional transition \(\tau_w\) that allows to model the waiting behavior.
It is, of course, possible that multiple robots are positioned in the same place. Since a firing transition is blocked during firing, it is impossible that multiple robots (> 1) perform the same action (moving, waiting). Therefore, we have to model this by using multiple waiting and moving transitions. Since the overall net is conservative (number of robot tokens do not change over time), the needed number of transitions for each place and movement is equal to the total number of robots. Fig. 3(b) shows only one waiting and one moving transition, the remaining ones are omitted for simplification.

Now, we want a task to be executed by one or several robots which takes a certain amount of time for execution. We model a task by a timed transition $\tau_{\gamma}$ with a duration of $d$ time units as shown in Fig. 4(a).

Places $p_{1,s}$ and $p_{1,f}$ reflect the status of the task, i.e., a token on $p_{1,s}$ marks the task as “ready to run” while $p_{1,f}$ states that the task has been executed. As a convention, we use the notation $\gamma$ (abbreviation for task) for each place and transition if they are task-specific. Further, a location-independent task is
modeled as a triple \((p_{\gamma i,s}^\gamma, \tau_{\gamma i}^\gamma, p_{\gamma i,f}^\gamma)\) where \(i\) is a unique task id, the index \(s\) marks the start place and \(f\) marks the task to be finished. It is easy to see that a task can be only executed exactly once. A task may be also location-dependent. In this case, the task is bound to a physical location at which it shall be executed and may require multiple robots (common execution). This requires two additional places that represent the physical locations (indicated as dashed places and arcs in Fig. 4(a)). The task then starts at the “left” place, ends at the “right” place and requires \(r\) robots for the execution. This modeling incorporates movement during execution. However, if the task does not incorporate movement (“left” place = “right” place), we only have to model one place.

If we further have dependencies between tasks, we can model this easily by using joint places as depicted in Fig. 4(b). There, we have three tasks \((1, 2, 3)\) represented by \((\tau_{\gamma 1}^\gamma, \tau_{\gamma 2}^\gamma, \tau_{\gamma 3}^\gamma)\) with individual execution times \((d_1, d_2, d_3)\) and location restrictions. Additionally, task 3 depends on the result of task 1 and 2. We model this by using the joint place \(p_j^\gamma\) which is a substitution for \(p_{\gamma 1,f}^\gamma, p_{\gamma 2,f}^\gamma\) and \(p_{\gamma 3,s}^\gamma\). After the execution of task 1 and 2 each of them places one token on \(p_j^\gamma\). So, we only have to mark the multiplicity of the arc \((p_j^\gamma, \tau_{\gamma 3}^\gamma)\) with 2 or, in general, with the number of preceding tasks.

Finally, we have to put both parts of the model together resulting in one TPN as shown in Fig. 5—due to the construction of our model elements, this composition is straightforward. Multiple waiting and movement transitions are omitted for clarity. In this example, we have in total four robots positioned on the physical grid. Two of the robots stay at the same physical location. Assuming \(r = 4\) requires the other two robots to move to the upper left place. If transition \(\tau_{\gamma 1}^\gamma\) fires, place \(p_{\gamma 1,f}^\gamma\) is marked and \(r\) tokens are put “back” on the grid. A marking of \(p_{\gamma 1,f}^\gamma\) represents the task has been executed successfully. During firing of \(\tau_{\gamma 1}^\gamma\), the \(r\) robots are involved in the execution (tokens are bound to \(\tau_{\gamma 1}^\gamma\)) and can not execute other
tasks at the same time. Since the execution of \( \tau_1 \) incorporates movement \( d_1 \) must be at least the time, or higher, required to move to the “right” location place directly. If \( r = 2 \), the task execution could start immediately (without “waiting” for the other two robots).

The overall goal is to execute all real-space-time tasks according to their requirements. This is represented in this model by a marking of the net in which all places \( p_{1,s}^\gamma \) of these tasks are marked. Additionally, the time required to reach such a marking starting at some initial marking should be minimized.

6. Finding an Optimal Schedule

Based on the model, we are now able to find an optimal schedule (minimal makespan) by translating the problem into a shortest path problem, which is solvable with the known methods of dynamic programming, cf., e.g., [26]. The straightforward solution is to spawn the state graph, label the graph’s edges with the time difference between states it connects, seek the state(s) that has the places \( p_{1,s}^\gamma \) marked for all tasks (goal states), and use dynamic programming [26] to find the shortest (in terms of time) path to a goal state. This path represents the optimal schedule.

In Section 4, we have defined the TPN’s state as \( s = (m, h) \). For our analysis, we use an extended state \( \xi \) that uses the time marking instead of the marking: \( \xi = (\mu, h) \). That enables us to apply the state equation given in [19] to calculate future markings:

\[
\mu_{i+\Delta t} = \mu_i \cdot Q^{\Delta t} + C \cdot \Psi_\sigma
\]

with

\[
\Psi_\sigma = \sum_{j=1}^{\Delta t} B^{(j)} \cdot Q^{\Delta t-j}
\]

Here, in correspondence to the classical state equation, \( C \) is the (timed) incidence matrix, \( \Psi_\sigma \) is the (timed) Parikh matrix, and \( B^{(j)} \) is the bag matrix (consisting of diagonal and zero submatrixes) for the \( j \)th step of the firing sequence \( \sigma \). For details and correctness of (10) and (11), please refer to [19].

With the help of (10) and (11), we can easily spawn the extended state space. All we have to do is to find the shortest path to a goal state, i.e., a state with a time marking

\[
\mu_x = \begin{pmatrix}
* & * & * & \ldots & p_1 \\
* & * & * & \ldots & p_2 \\
* & * & * & \ldots & p_3 \\
* & * & * & \ldots & p_4 \\
* & * & * & \ldots & p_{1,s}^\gamma \\
1 & * & * & \ldots & p_{1,f}^\gamma \\
\end{pmatrix}
\]

In the marking \( \mu_x \), elements that are declared with a \#-symbol do not matter, it is only important that \( p_{1,f}^\gamma \) is marked.

However, the state space of a TPN might be infinite, even if the Petri net itself is finite. In addition, it might be possible that no goal state is reachable. Thus, in the following we are discussing both aspects.
We start with the reachability. For our specific model, we distinguish two kinds of non-reachability:

- Non-reachability because of lack of resources;
- Non-reachability because of structural problems, i.e., circular task dependencies or tasks in disconnected areas of the topology.

Considering (non-)reachability of the first kind. Let \( r_{\text{max}} = \max(r_i) \) and \( n_\gamma \) the number of tasks. Obviously, no goal state can be reached if \(|\mu_0| < n_\gamma + r_{\text{max}}\). In this case, there is 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.

To find non-reachability of the second kind, the state equation (10) can be used, c.f. [19]. However, this kind of non-reachability can already be excluded at model time: It is easy to see that no non-reachability of the second kind will occur, if there are no circular dependencies between tasks and the topology is strongly connected. We assume, that both is checked at modeling time by the usual methods.

Regarding the possible infiniteness of the state graph, consider the model component nets as described in Section 5. Each of them is conservative with respect to the timed marking. I.e.,

\[
\forall \mu, |\mu| = \sum_{i=1}^{P} \sum_{j=1}^{d} \mu_{i,j} = \text{const} \quad (13)
\]

Also, the composition of the components keeps the conservativeness. Thus, the space of time markings in each model is finite. In addition, each clock vector \( h(\tau) \) can only have the values \( 0, \ldots, D(\tau) \), i.e., the clock vector state space is finite. Thus, the overall state space is finite, too.

By the method described above, we are able to find an optimal schedule. It corresponds to the shortest path to a goal state, that can be found by the usual algorithms, e.g., Dijkstra’s algorithm [7]. Relaxation corresponds to joining of two firing sequences, c.f. Fig. 6.

The complexity depends on the spawning of the state space, that has an exponential complexity, and the shortest path algorithm, that has a polynomial (or better, depending on the actual algorithm)

\[\text{Obviously in the timeless net; in the timed net because of } \tau_w \text{ as loop at all location places.}\]

\[\text{Please note, that the net is not conservative with respect to the marking.}\]
complexity in the number of states. Clearly, the complexity of the overall method is dominated by the spawning of the state space.

However, there exists an approach to improve the runtime in many cases: The shortest path algorithm can be executed along with the spawning of the state space. Then, spawning can be aborted, when a goal state is found, and all other paths have an equal or higher collected weight (distance). Since all edge weights have a positive weight (time), the path to the found state is an optimal solution.

7. Case study

In order to evaluate our model, we performed a case study. This case study is based on our robot testbed consisting of several taskit Robot1-0 as depicted in Fig. 7. The robots are equipped with an ARM926EJ-S processor (ARMv5) running at 400 MHz and have 64 MB SDRAM. For sensory, each robot features $6 \times$ ultrasound, $13 \times$ infrared and two RGB LEDs. They are able to communicate using WLAN and execute a swarm operating system developed by us [10]. These robots operate on a playground measuring $2 \times 2$ meters and are able to perform movement and observation actions. For physical location of the robots we use a Microsoft Kinect and transfer location data wirelessly to the robots.

For modeling this case study, we discretized the playground based on the robot size into a $12 \times 12$ grid. In order to get actual models, we implemented a generator tool that is able to create TPN models of such problem instances by composing model elements as described in Section 5. It is possible to specify an $n \times m$ grid, number and positions of robots and tasks (including dependencies) that can be attached to physical places. Durations for movement transitions can be edited manually or a distribution of transition durations can be applied in order to automatically generate topologies. The generator outputs files in the pnt and tmd format that we use as input into INA (Integrated Net Analyzer) [16] for model checking and analysis. INA can find the shortest path in a state space. However, INA does not apply the improvement discussed at the end of Section 6.

Since our approach has an exponential worst-case complexity, we limited our test study to rather small test cases. We modeled our problem as follows: We have two robots dealing with five tasks. Three tasks are coupled by a precedence, the other tasks are independent. Four tasks need one robot for execution, the remaining one requires two robots. The execution locations of these tasks as well as the initial positions of the robots are generated randomly. We assume a random terrain defined by the
following distribution for movement transitions: 50% of the transitions have 1 time unit, 12.5% have 2 and another 12.5% have 3 time units. The remaining 25% simulates obstacles indicating no possible movement.

We executed 1000 experiments. A modern computer requires approximately one to three hours runtime for a single run in average on a single core, what prohibits the use of our approach as an on-line schedule. However, for an off-line method to establish e.g., a production schedule, this number seems reasonable. In addition, todays microprocessors consist of multiple cores per single chip, and our approach is well suited for parallelization. Unfortunately, INA is not implemented to cope with multiple cores, so parallel execution was not possible for our case study.

Beside of the correctness of the result, we were interested in the number of states in dependency on the actual configuration.

More precisely, the state space depends on the grid size, the time conditions for movements in the grid and the positioning of tasks and robots. Our results are shown in Fig. 8 and 9. Fig. 8 shows the amount of generated states for the reachability graph in dependence of the size of the grid by 2 robots and 1 task. Fig. 9 shows the distribution of the states for the full configuration.

8. 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. If a task has to be executed jointly by multiple robots and is bound to a 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. The core of the presented work here is to find an optimal space-time schedule for mobile resources where optimal is defined as minimal makespan.

In order to deal with these problems, an appropriate modeling technique is needed. In this paper, we propose 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 TPN 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: First, we want to weaken the restrictions given in Sect. 3 by allowing both 3D movements restrictions on time and space. 3D can be simple added to the model by including a third dimension into the grid while time restrictions can be modeled in a similar way as presented in [21] by adding timed transitions that mark special places in case they finish firing earlier than the transition modeling the execution of the appropriate task. Next, 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 suspending and resuming the execution of a task. Finally, we aim to define and describe a general approach to space-time scheduling which is based on a continuous space and may include deadlines and, e.g., specific formations of robots.
References


