Profinet IO IRT Message Scheduling

Zdeněk Hanzálek, Pavel Burget, Přemysl Šůcha
Czech Technical University in Prague, Faculty of Electrical Engineering
Department of Control Engineering, Centre for Applied Cybernetics
{hanzalek,burgetpa,suchap}@fel.cvut.cz

***Abstract*—This paper presents an algorithm that allows one to create a static schedule of the Profinet IO IRT communication, which is an industrial Ethernet protocol standardised in IEC 61158. This algorithm offers an alternative to the available commercial tool, providing comparable results regarding the resulting time schedule length. Furthermore, we extend the problem by useful time constraints providing a greater flexibility with respect to the individual messages. Due to this flexibility, it is possible to place the selected messages in various parts of the communication cycle, to define end-to-end delays, or to increase the computational time available for the main-controller application, for example. The solution is based on a formulation of the Profinet IO IRT scheduling problem in terms of the Resource Constrained Project Scheduling with Temporal Constraints ($PS\,|temp|\,C_{max}$).**

I. Introduction

Contrary to the original intended use of the Ethernet, there has been a demand to use this medium even at the *field level* (real-time communication with sensors and actuators) of industrial automation. The intention is to have a single network technology, going through the whole factory up to the office level. One significant fact about the Ethernet and its usage for real-time control is that it is wide-spread and supported by the chip manufacturers and thus intensively focused by the industry.

The field level has been traditionally occupied by fieldbus systems (e.g. Profibus is one of the well-established industrial standards, having been addressed by the research community [1] or [2]). Some of them are based on time-triggered approach, known for its predictable behavior and composability (see [3], [4] and [5]). Several authors investigated adaptive behavior of time-triggered networks (TDMA with Slot-Skipping [6], FTT-CAN [7] and FlexRay [8]). There have been numerous attempts examining the possibilities for the Ethernet to enter the domain of fieldbuses (see survey articles [9], [10]) and there have been many protocols proposed by academia and industry. However the Ethernet was not designed to satisfy the requirements of real-time communication [11], [10]). Therefore many contributions from the research community suggested solutions how to overcome the non-deterministic behaviour of the Ethernet medium access and how to provide real-time communication guarantees. Some of the suggested methods propose traffic smoothing [12], centralised medium access [11] or decentralised access [13]. The traffic smoothing relies on the fact that, up to a certain communication load, no collisions occur on the medium with a high probability. Thus, the generation of the messages by the application layer must be controlled in order to keep a low communication load. The centralised medium access may rely on the presence of a manager node (master) that determines when the individual controlled nodes (slaves) should transmit their messages, and ensures that the real-time data is timely separated from the non-real-time data. The decentralised access relies on the assumption that every node knows its schedule, defining when to start the communication.

The above mentioned approaches, which satisfy the real-time requirements with Ethernet, have been realised in industrial standards, such as DDS [14], Ethernet Powerlink [15] or Profinet IO [16], [17]. The latter one, Profinet IO (specifically its RT Class 3, i.e. *Isochronous Real Time – IRT*) is dealt with in the rest of the paper and thus we discuss some other works related to it here.

Profinet IO IRT is a hard-real time communication protocol, which uses static schedules for time-critical data to fulfil the requirement on timeliness of the data delivery. The individual-node schedules are downloaded into the nodes, each of which contains a switch. Thus, a special hardware (switch) is required to be able to accept the respective schedule. There is a commercial tool, being part of the Siemens[1] Simatic Manager professional engineering software [18], that is able to generate the schedules. However, no papers describing the scheduling algorithm used in the commercial tool have been published. There is just some related work such as [19], where graph algorithms used to solve some scheduling or planning problems are described, but they are not directly connected to the Profinet IO IRT scheduling. Paper [20] describes a scheduling algorithm that is based on colouring the edges of a graph that represents the network topology, but there are certain limitations, such as no link-delay or bridge-delay parameters are taken into consideration, and only the half-duplex links are assumed.

The rest of the paper is organized as follows. Section II provides a brief review of the Profinet IO IRT standard and discusses the necessity to use special hardware (Ethernet switches). Furthermore, it shows us how to derive the input timing parameters in order to obtain a message schedule, which can be executed on the Profinet IO IRT hardware. Section III refines the Profinet IO IRT scheduling problem including the extension by release dates, deadlines and required end-to-end delays of the messages. The main contribution of

[1]Siemens, Simatic, Simotion and Sinamics are registered trademarks of Siemens

the paper is shown in Section IV, presenting a solution of the problem. The scheduling algorithm first takes the parameters of the topology and messages and formulates an instance of off-line scheduling $PS\,|temp|\,C_{max}$, while creating a graph of tasks. In the graph, each unicast message is a chain of tasks executed on communication links starting at a source node and ending at a destination node, and each multicast message is a tree of such tasks. The problem represented by the graph is further handled by our solver (optimal or heuristic one, up to the size of instance). Section V illustrates how the problem extension (release dates, deadlines and required end-to-end delays of messages) may be conveniently used in the application-design phase. Section VI reports on the evaluation of our design tool which is currently tested by a Profinet test laboratory. Based on benchmarks, with up to 1000 tasks, we compare the schedule length and the algorithm computation time of our tool with the ones achieved by the commercial tool [18]. Section VII concludes the paper and offers some suggestions for future work.

II. Profinet IO Industrial Protocol

Profinet IO is defined in [16] and [17] as part of the international IEC 61158 standard and one of its aims has been to replace traditional fieldbus systems at the field and cell levels in the control system hierarchy. As there are many application areas with different requirements there are also various classes of service defined in Profinet IO. In fact, there are 4 of them, called RT Class UDP, and RT Class 1 to RT Class 3, which differ in the real-time capabilities and the availability of the clock synchronisation. For the application with softer real-time requirements, the basic communication principle relies on a switched full-duplex topology where messages are forwarded according to the forwarding rules defined within IEEE 802 (see [21], [22], [23] or [24]).

The process data are exchanged between the *IO Controller* and *IO Device* using one of the specified real-time classes. There can also be *non-real-time – NRT* (having lower priority than RT Class 1 and RT Class UDP) data such as diagnostics or configuration that is transmitted using the Profinet IO protocol, or other data using various protocols based on the TCP/IP suite. In-depth information about Profinet IO can be found in the IEC 61158 standard [16], [17], or in [25].

A typical topology is depicted in Figure 1, which shows four connected nodes. Such a topology supposes there is a switch integrated in each node, i. e. for Figure 1, there are 4-port and 2-port communication nodes. Each communication

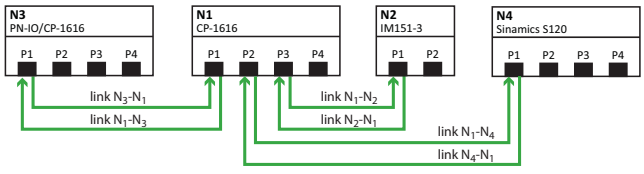

Fig. 1. Topology with 4 nodes

link between two communication nodes is shown as two one-way links, which corresponds to the full-duplex type of communication.

Figure 2 shows the *Profinet IO communication cycle* (also called communication period or send clock), divided into individual *communication intervals*. For the purpose of this paper, the *red interval*[2] is important as it contains the *RT Class 3* data that is forwarded according to a static communication schedule. The RT Class 3 communication is the highest-priority one, and is required to be strictly isochronous. It means the individual data frames have to be transmitted at time instants, which are equidistant with respect to the consecutive communication cycles. This type of communication, also called *time-triggered*, relies on the ability of individual nodes to send the respective messages exactly at the predetermined time instants. To be able to accomplish it, the nodes' clocks must be synchronised with such a precision that allows the jitter of the communication-cycle length to be as low as possible, whenever the *Precision Transparent Clock Protocol (PTCP)*, as defined in IEC 61158, is used. In a typical scenario, the synchronisation frames are sent in every communication cycle and are subject to the static schedule.

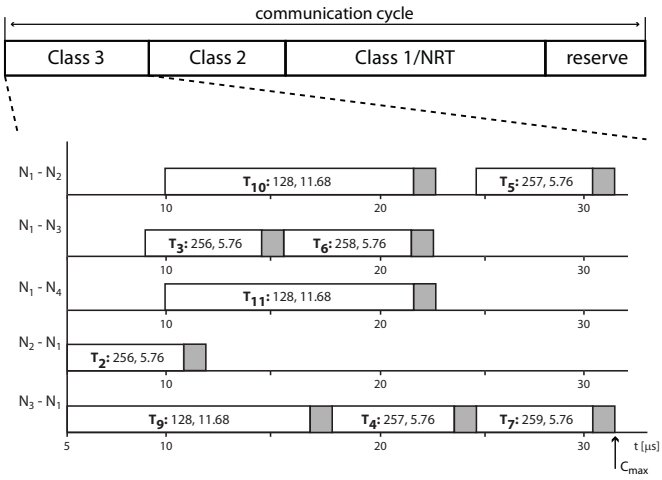

Fig. 2. Profinet IO communication cycle

The schedule is computed during the engineering phase and loaded into the individual nodes. An example schedule is represented by a Gantt diagram in the lower part of Figure 2, where each line corresponds to the link used and each task is labeled by its name, message ID and transmission delay in μs (inter-message gaps are represented by gray rectangles). The schedule breaks the standard forwarding rules intentionally to ensure that no queuing delays occur in the switches (remember that a special switch is integrated in each node for RT Class 3 communication. No communication other than the scheduled one is allowed. If it occurred, it would be blocked by the respective switches.

[2]The expression *red interval* is defined in [16] and means the communication interval dedicated for the RT Class 3 communication. As such, it is used in this paper.

A. Scheduling Problem Input Parameters

Each switch introduces a considerable delay, composed from port-dependent and bridge-dependent parameters. An in-depth description of the delay parameters is provided in [16]. The main points, which are crucial for the specification of the scheduling problem, are presented here.

A line is a communication connection between two nodes and the line delay consists not only from the cable delay but also the transmit-port delay and the receive-port delay must be taken into account. Another important parameter is the bridge delay because it represents the time, which is needed for a packet to be processed in the bridge. The packet is processed either locally, i. e. the local device is the recipient of the packet, or it is resent to another port. In the latter case the *cut-through* mechanism or the *store-and-forward* mechanism is used as it depends on how the schedule for individual messages looks, as described later. Thus, the *communication line delay* T_{LD} between two ports is given as

$$T_{LD} = T_{TxD} + T_{CD} + T_{RxD} + T_{ad} + T_{BD}, \qquad (1)$$

where $T_{CD} = T_{PD} \cdot L$. Here, T_{PD} is the single-bit propagation delay per one meter and L is the cable length given in meters. The value of T_{PD} ranges usually from 4 to 6 ns per one meter of cable, the typical value is 5 ns/m. The meaning of the other delay parameters is as follows: T_{TxD} is the transmit-port delay , T_{RxD} is the receive-port delay, T_{BD} is the bridge delay. The additional link delay parameter T_{ad} provides a safety margin for the clock synchronisation precision.

The delay parameters for the topology in Figure 1 are given in Table I. They come out from a real installation based on Profinet IO nodes, namely from the device data sheets (GSD files of CP-1616, IM151-3 and Sinamics S120). Based on Table I and Equation (1), Table II can be computed to produce

node	T_{TxD}[ns]	T_{RxD}[ns]	T_{BD}[ns]
N1	1192	363	1720
N2	158	350	2720
N3	1192	363	1720
N4	1212	418	1920

TABLE I
DELAY PARAMETERS RELATING TO FIGURE 1

the resulting link delays for each of the links in the network. For the sake of simplicity, we take all links to be 100 m long. This results in $T_{CD} = 600$ ns if $T_{PD} = 6$ ns. The *additional link delay* is 1 μs as described in [16].

In order to be able to generate the message schedule, the messages to be communicated must be known *a priori*. For our example, the messages are given in Table III. The table contains messages with *process data* (message IDs from 256 to 259), and a *synchronisation* message (having message ID 128). Of course, only the RT Class 3 communication, i. e. messages are in the red interval, is taken into account.

link	T_{LD} [ns]
$N_1 \rightarrow N_3$	4875
$N_1 \rightarrow N_4$	5130
$N_1 \rightarrow N_2$	5862
$N_2 \rightarrow N_1$	3841
$N_3 \rightarrow N_1$	4875
$N_4 \rightarrow N_1$	4895

TABLE II
LINK DELAYS RELATING TO FIGURE 1

ID	path	T_{TD} [ns]	$\widetilde{r}$ [ns]	$\widetilde{d}$ [ns]	$\widetilde{e}$ [ns]
256	$N_2 \rightarrow N_3$	5760	5000	20000	11000
257	$N_3 \rightarrow N_2$	5760	15000	40000	15000
258	$N_1 \rightarrow N_3$	5760	15000	–	–
259	$N_3 \rightarrow N_1$	5760	20000	35000	–
128	$N_3 \rightarrow \{N_1,N_2,N_4\}$	11680	5000	{–,–,–}	{–,17675,17675}

TABLE III
MESSAGES FOR THE NETWORK IN FIGURE 1

The length of the messages is given by the application. The length of the synchronisation information is 114 bytes and let us assume that the length of the process data is 40 bytes.

In addition, a further 32 bytes are added to form the Ethernet-frame head and tail [23]. Therefore we expect the synchronisation messages to be 146 bytes long and the process data messages to be 72 bytes long.

The *message transmission delay* T_{TD} is the time to transmit all the bits of a message from the transmit queue to the communication line. $T_{TD} = l \cdot b_{TD}$, where l is the message length in number of bits, and b_{TD} is the *single-bit transmission delay*, which is 10 ns in our case.

The schedule, resulting from the parameters in Tables II and III, is shown in Figure 2. It can be seen that N_3 acts as clock master as it is the source node of the synchronisation message denoted as 128. This message is also an example of the multicast communication because it is forwarded by every switch that receives it. Other messages are unicast, for example message 256 is forwarded by N_1. This is because of the peer-to-peer communication in the switch-based topology. Columns $\widetilde{r}$, $\widetilde{d}$ and $\widetilde{e}$ represent the constraints required by the application engineer and are explained in Section III.

III. PROBLEM REFINEMENT

The objective of the paper is to find a time-triggered message schedule for the red interval (see Figure 2), which is based on a list of messages created in the engineering system, and on the network topology composed of individual switches and communication links among them as described in the previous section. To be able to propose the scheduling algorithm, we formalise the scheduling problem later on, while keeping the following rules:

1) The maximum length of the red interval, given by the application requirements, defines the upper limit

by which the communication must be finished. At the beginning of the red interval there is a *safety margin* of $5\,\mu s$ (see [16]).

2) The messages on the same link are separated with a minimum inter-message gap, which is at least 960 ns (see [23]) but a commonly used value is 1120 ns (see [18]). This inter-message gap is added to T_{TD} for the message-schedule computations.
3) As soon as the first bit of a message is received on a port, i. e. T_{LD} after it was sent from the previous node, and the message is to be forwarded to another link, it is forwarded. The *cut-through mechanism* describes such a situation and can be seen on message 256 in Figure 2. The cut-through mechanism can not be used when the outgoing port is busy (the schedule for the respective port says so). In such a case, the message is stored in the switch message queue and the (partial) *store-and-forward* mechanism describes it as demonstrated on message 257.
4) If the link delay T_{LD} is greater than the message transmission time T_{TD}, the respective message will be processed by one communication node. However, if $T_{LD} < T_{TD}$, two nodes may process a different part of the same message at the same time.
5) The synchronisation message does not have to be sent at the beginning of the interval. It can be planned anywhere within the interval if the synchronisation message arrives at a device at equidistant time instants.

Some messages may be required to be sent in specific parts of the red interval in order to prolong the computation time available for the controller application (further illustrated in Section VI).

Therefore, we extend the scheduling problem by the message *release date* $\widetilde{r}$ (specifying the earliest moment the first bit of the message is sent from the source node), the message *deadline* $\widetilde{d}$ (specifying the latest moment the last bit of the message is received by the destination node), and the required message *end-to-end delay* $\widetilde{e}$ (specifying the time elapsed from the moment the first bit of the message was sent from the source node, to the moment the last bit of the message is received by the destination node). Please note that this definition of the end-to-end delay does not involve the processing time in the source and destination nodes.

IV. Solution of Profinet IO IRT scheduling

This section describes an algorithm which finds a schedule of messages on communication links in a time-triggered network while minimising the schedule length and respecting all temporal and resource constraints. Each unicast message may be seen as a chain of tasks executed on communication links starting at the source node and ending at the destination node. In a similar way, each multicast message is seen as a tree of tasks. As a result, each port of a node contains a list of tasks to be processed and for each task there is the start time, i. e. an instant, when the message is to be sent. This principle is used in Profinet IO as described in the previous text and there is a commercial design tool [18] deriving such start times. Due to the tree topology of the Profinet IO communicating nodes, the rooting of messages is determined, thus the rooting is not subject to the discussed optimisation and each task is assigned to a dedicated link. The objective of this work is to make a documented optimisation algorithm, which is further able to incorporate various constraints, thus being able to solve more complex problems than the algorithm being part of [18]. The key idea is to formulate the Profinet IO IRT scheduling problem in terms of the Resource Constrained Project Scheduling with Temporal Constraints minimizing the schedule makespan (denoted $PS\,|temp|\,C_{max}$ in off-line scheduling terminology), so that a possible user is not bound to a particular implementation but he/she can choose from a variety of algorithms solving this problem. Therefore, the scheduling algorithm first generates the instance of $PS\,|temp|\,C_{max}$ from the input data (such as the ones in Tables I and III), then the problem is solved and finally a result is interpreted in terms of the start times of the tasks executed on the communication links.

A. $PS\,|temp|\,C_{max}$ problem

Traditional off–line scheduling algorithms [26] typically assume that deadlines are absolute, i. e. the deadlines are related to the beginning of the schedule. On the other hand, when assuming, for example, the required end-to-end delay of the message in a time-triggered network, we may conveniently represent a given timing requirement by a deadline of task T_j related to the start time of task T_i. In such a case, the absolute deadlines cannot be calculated *a priori*, and therefore we use "relative" deadlines.

The concept of precedence delays and "relative" deadlines, called *(general) temporal constraints* or *minimum and maximum time lags*, have been introduced by Roy [27] and further developed by Brucker et.al. [28]. Classified as $PS\,|temp|\,C_{max}$, it is a widely studied problem of a resource constrained project scheduling problem (so called limited renewable resources [29]) with temporal constraints (also called time windows [30]) and makespan minimisation. The problem was studied by the operations research community, but similar principles also appeared in optimisation of compilers for multiprocessor machines [31] and in symbolic representation of states in timed automata [32].

The set of n tasks $\mathcal{T} = \{T_1, \dots T_i, \dots T_n\}$ with temporal constraints is given by a task-on-vertex graph G [27]. A message transmission over the communication link corresponding to task T_i is represented by vertex T_i on graph G and has a non-negative duration corresponding to p_i, the processing time of the task. The scheduling problem is to find a *feasible schedule* $(s_1, s_2, \dots s_n)$, satisfying the *temporal constraints* and *resource constraints*, while minimising the makespan C_{max}. The value of C_{max} corresponds to the length of the red interval in the case of Profinet IO.

Temporal constraints between the vertices are represented by a set of directed edges. Each edge from vertex T_i to vertex T_j is labeled by a weight w_{ij} which constrains start times of

the tasks T_i and T_j by the inequality

$$s_j - s_i \geq w_{ij}. \tag{2}$$

There are two kinds of edges: the edges with positive weights and the edges with negative weights, but Equation (2) holds for both of them. The edge with a positive weight w_{ij} (giving minimum time lag), indicates that s_j, the start time of T_j, must be at least w_{ij} time units after s_i, the start time of T_i. We use the positive weights w_{ij} to represent:

- *precedence relation* ($w_{ij} = p_i$, i. e. T_j starts when T_i is completed at the earliest) is used for example, when the output product of T_i serves as the input of T_j
- *delayed precedence relation* ($w_{ij} \geq p_i$, i. e. T_j starts $w_{ij} - p_i$ time units after completion of T_i at the earliest) is used for example, when T_i represents a painting operation, $w_{ij} - p_i$ represents quarantine time needed to become dry and T_j represents a packaging operation (we used similar model to represent pipelined arithmetic unit on FPGAs [33])
- *overlapping precedence relation* ($w_{ij} \leq p_i$, i. e. T_j starts $p_i - w_{ij}$ time units before completion of T_i at the earliest) is used for example, when the Profinet node uses a cut-through mechanism (e.g. communication of message 256 over the network in Figure 1 starts transmission on line N1-N3 before reception on line N2-N1 is completed)
- *release date* $\widetilde{r}_j$ of task T_j when $s_i = 0$.

The edge, from vertex T_j to vertex T_i with a negative weight w_{ji} (giving the maximum time lag), indicates that s_j must be no more than $|w_{ji}|$ time units after s_i. Therefore, each negative weight w_{ji} represents $\widetilde{d}_j(s_i)$, a deadline of T_j depending on s_i, such that $\widetilde{d}_j(s_i) = s_i + |w_{ji}| + p_j$. Consequently:

- when T_j is the last task of the message and T_i is the first task of the same message, the edge with a negative weight may be conveniently used to represent the required end-to-end delay $\widetilde{e}_{ji} = |w_{ji}| + p_j$
- when $s_i = 0$ (i. e. T_i is the task scheduled at time 0), the edge with a negative weight may be conveniently used to represent the absolute deadline $\widetilde{d}_j = |w_{ji}| + p_j$.

Some other constraints that occur frequently in practice can be modeled by temporal constraints as follows:

- *fixed start time* of task T_j precisely at time t_j, when $s_i = 0$, $w_{ij} = t_i$ and $w_{ji} = -t_i$
- *simultaneous start* of tasks T_i and T_j, when $w_{ij} = w_{ji} = 0$
- *simultaneous completion* of tasks T_i and T_j, when $p_i \geq p_j$ and $w_{ij} = -w_{ji} = p_i - p_j$
- *consecutive execution* of tasks T_i and T_j without any delay in between, when $w_{ij} = p_i$ and $w_{ji} = -p_i$.

The non existence of a cycle with positive length is a necessary condition for the existence of a feasible schedule. Moreover, the solving algorithms have to consider the *resource constraints* preserving two tasks to be executed on the same resource (communication links in our case) at once. From the time complexity point of view, the problem is NP-hard, due to the resource constraints, which leads to a non-convex representation of the set of feasible solutions. Therefore, the optimal solutions are typically found for up to one hundred tasks ([34],[33]), depending on the number of tasks conflicting on one resource. Heuristic algorithms (see survey in [35] or our extension by semaphores in [36]), attractive for industrial sized problems, are able to handle up to one thousand tasks in few seconds.

B. Algorithm Description

The first step of the algorithm may be illustrated as a construction of the oriented graph G in Figure 3 from the data in Tables II and III. Each communication link between two nodes is taken as one resource, which can execute the tasks to be scheduled. A task execution corresponds to a transmission of a message on the respective link. The unicast message corresponds to a chain of tasks and the multicast message corresponds to a tree of tasks.

1) Unicast Messages: Vertex T_2 in graph G in Figure 3, representing task T_2, corresponds to the transmission of message 256 over line 2-1 and T_3 corresponds to the transmission of the same message over line 1-3. In a similar way, T_4 and T_5 correspond to the transmission of message 257 over lines 3-1 and 1-2, T_6 corresponds to message 258 and finally T_7 corresponds to message 259. The dummy task T_1, preceding all other tasks, corresponds to the event which indicates the start of the schedule, since the C_{max} minimisation always schedules task T_1 at time 0. Tasks $T_2 \ldots T_7$ are executed on the corresponding lines and task T_1 is executed on a dummy resource. The processing time of tasks are $p_2 = \ldots = p_7 = 6880$ and $p_1 = 0$. To remind us where these values come from, let us have a look at Table III for the message durations, and Table II for the link delays. In addition to the message transmission delay T_{TD}, each processing time also includes the inter–message time in order to keep the network idle and not to schedule the subsequent message at that time. All unicast messages, in this example, have the same transmission delay of 5760 ns and the inter-message time is 1120 ns.

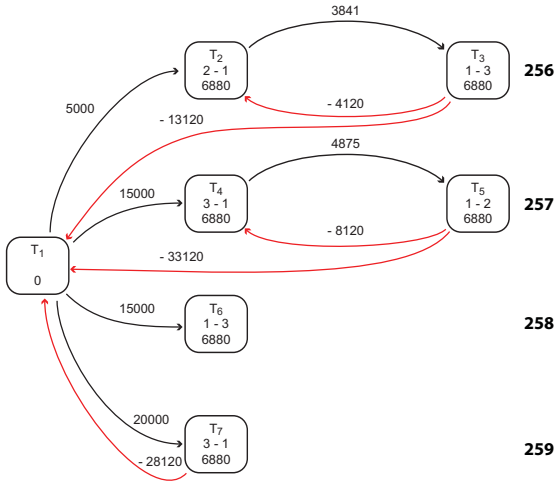

Fig. 3. Unicast messages graph for Fig. 1

The line delay is represented as an edge with positive weight T_{LD} interconnecting the tasks of the given message in the chain corresponding to the path from the destination to the source node. The release date of a message is represented as an edge with weight $\widetilde{r}_{ID}$ from T_1 to the first task of the message. The deadline of a message $\widetilde{d}_j = |w_{ji}| + p_j$ is represented as an edge with weight $-(\widetilde{d}_{ID} - p_j)$ from T_j, the last task of the message, to T_1 scheduled at time 0. The required end-to-end delay of a message is represented as an edge with weight $-(\widetilde{e}_{ID} - p_j)$ from T_j, the last task of the message, to T_i, the first task of the message.

2) Multicast Messages: Graph G in Figure 3 does not include the synchronisation message 128, which must be present in the final schedule. This is a *multicast message* and it is defined by the source node N_3 and a set of the destination nodes $\{N_1, N_2, N_4\}$. The graph of this multicast message, shown in Figure 4, should be interpreted as an extension of the unicast messages in Figure 3.

The multicast tree of message 128 consists of the tasks T_8, T_9, T_{10}, T_{11}. Task T_8 is the dummy root of the tree and it is joined to T_1 which has been created in Figure 3 already. Each link on the path from the source node to the destination node in Figure 1 corresponds to the task on the path form the root task to the leaf task in Figure 4. The release date of a multicast message is represented as an edge from T_1 to T_8. Deadlines or required end-to-end delays of the destination nodes are constructed while using the edges with negative weights from the corresponding task to T_1, in the case of $\widetilde{d}$, or to T_8 in the case of $\widetilde{e}$, respectively.

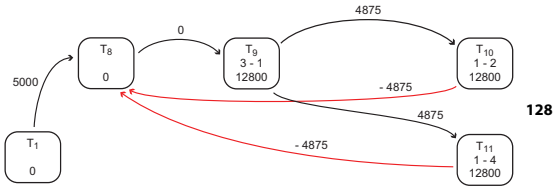

Fig. 4. Extension of Fig. 3 for multicast

The requirement to forward the synchronisation message promptly by each switch may be expressed by the required end-to-end delay $\widetilde{e}$ related to each destination vertex of the multicast message while using an edge with negative weight.

Graph G representing both unicast and multicast messages is given by W, the adjacency matrix of weights $w_{i,j}$:

$$\begin{array}{cccccccccc}
-\infty & 5000 & -\infty & 15000 & -\infty & 15000 & 20000 & 5000 & -\infty & -\infty & -\infty \\
-\infty & -\infty & 3841 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-13120 & -4120 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & 4875 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-33120 & -\infty & -\infty & -8120 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-28120 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 4875 & 4875 \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -4875 & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -4875 & -\infty & -\infty & -\infty
\end{array}$$

In W, $w_{i,j} = -\infty$ if there is no edge in G from T_i to T_j.

Each feasible schedule has to satisfy the resource constraints (at most one message is transmitted over the communication line at the given time) and temporal constraints (Equation 2 holds for all weights in adjacency matrix W).

Therefore, the scheduling algorithm, which takes the parameters in Tables II and III as the input and generates the vector of start times s as the output, consists of two phases. The first phase takes the algorithm inputs and creates graph G characterized by processing time of the tasks p, incidence matrix W and assignment of tasks to links a. The second phase, being the solver of the $PS\,|temp|\,C_{max}$ problem, takes p, W, a and finds s. The resulting schedule of the example is shown in Figure 2, while omitting the dummy tasks.

V. PROFINET IO APPLICATION POINT OF VIEW

This section shows how the parameters $\widetilde{r}$, $\widetilde{d}$ and $\widetilde{e}$ can be used by an application engineer to influence the timing of the messages with respect to the application demands. We describe two ways how this influence of the timing can be used: (1) prolongation of the computation time available for the controller application (see Figure 5), and (2) specification of the end-to-end delays for individual messages. However, other application requirements can be fulfilled by formulating them in terms of the message scheduling as shown in Section IV. Some experiments with numerical results are shown in Section VI.

A. Controller Application

Figure 5 shows another view on the communication cycle T_{DC}. An input-data frame is sent from an IO device to an IO controller, where the *controller application* computes the output data and the controller sends an output-data frame back to the device. The controller application runs with a constant period to be able to compute and issue the control action at equidistant time instants. Together, with the background tasks, the controller application forms the *controller application cycle (CAPC)* T_{CAPC}. In our case $T_{CAPC} = T_{DC}$. The application starts at the time instant T_{AS} after the beginning of the communication cycle, and finishes at time T_{AE}, which is before the end of the communication cycle T_{DC}. The input data must be available at the controller application in the current CAPC in order for the application to be able to send the computed output data during the next data cycle to affect the controlled process as soon as possible. It means that the input data must arrive and get prepared in the application input buffer some time in the interval of $\langle 0\,;\, T_{AS}\rangle$. Similarly, the control action represented by the output data must be computed and issued before the end of the communication cycle, i. e. some time in the interval of $\langle T_{AE}\,;T_{DC}\rangle$.[3] From the IO-device point of view, there is also a local application in the device, performing the local control tasks. The current state of the inputs is transmitted to the controller application in the form of a frame with input data, and the control action is based on an output-data frame received from the controller application via the network. In order for the *input-output delay* $\langle$send inputs $\rightarrow$ computation $\rightarrow$ receive outputs$\rangle$ to be as short as possible, the input data must be ready to be sent T_I before

[3]The bold arrows in Figure 5 indicate the relating movement of the output-data frame and prolongation of the computation time available for the controller application.

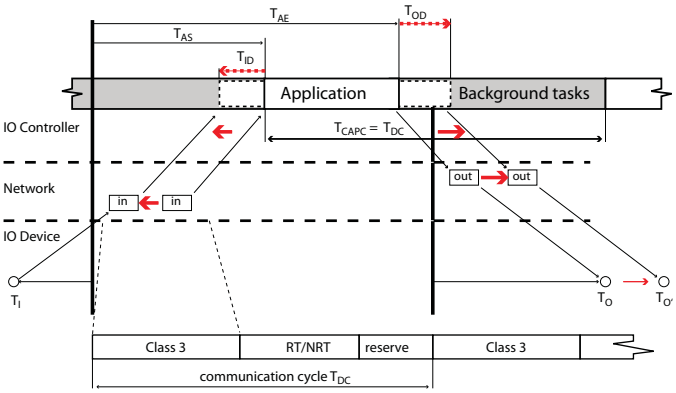

Fig. 5. Controller Application Timing

the start of the communication cycle. Similarly, the output data must be ready in the IO device at T_O in order for it to be copied from the output buffer to the actual outputs. Thus, the end-to-end delay $T_{IO} = T_I + T_{DC} + T_O$.

Looking back at Figure 5, we see there are bold arrows signalling a possible prolongation of the time available for the controller application. Thus, the application start time becomes $T'_{AS} = T_{AS} - T_{ID}$, and the application end time becomes $T'_{AE} = T_{AE} + T_{OD}$. In such a way, the cycle time does not change but the time the application is provided with, increases by $(T_{ID} + T_{OD})$. The new application start T'_{AS} brings a new requirement: all the necessary input data must be in the application input buffer earlier than T'_{AS}. Similarly, due to T'_{AE} all the output data must not be sent before the application finishes. This requirement may be accomplished by moving the input-data messages into an earlier part of the red interval (given by $\widetilde{d}$), while the output-data messages must be moved to its later part (given by $\widetilde{r}$). Thus, the output-writing point T_O becomes T'_O whereas the input-reading point T_I does not move. This feature is the contribution of this paper and as such is not described in the IEC 61158 standard.

B. Synchronisation Messages

The scheduling of the synchronisation messages can utilise the end-to-end delay parameters in order for the messages to be forwarded without any additional delay. In the case of a hardware-based time stamping, which inserts the actual time value in the outgoing message, the delay is measured autonomously and is capable to adapt to any additional hold-up in the switch. However, the time-stamping mechanism is not standardised and we cannot rely on it. Thus, for safety reasons, we want the synchronisation message to be forwarded using the cut-through mechanism in every device. The instant delay necessary to recognise the destination is hidden in the T_{BD} parameter, which T_{LD} is composed of. Thus, the end-to-end delay values for each synchronisation message ensure no hold-up occurs.

VI. Experimental Results

The scheduling algorithm, presented above, has been tested on network topologies created in the professional engineering tool [18], which is used among other things for the Profinet IO network configuration. A sample topology of a network for motion control applications is depicted in Figure 6. There

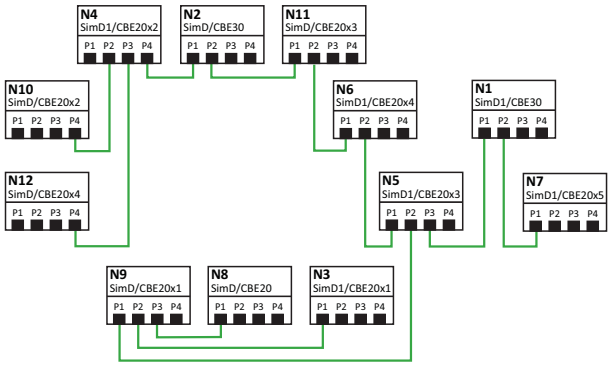

Fig. 6. A topology to show the communication schedule

are two motion controllers Simotion D (nodes N_1 and N_2), acting as Profinet IO controllers, and ten Sinamics drives, acting as Profinet IO devices. A message schedule that was generated by our design tool based on the the scheduling algorithm presented in Subsection IV-B, is shown in Figure 7. The input-data messages are filled with a gray colour, output-data messages are short white boxes and the synchronisation messages are long white boxes. It can be seen that both the input and output-data messages are mixed throughout the interval, i. e. there have been no constraints put on the placement of the respective messages in the interval.

A schedule with the message release dates, deadlines and end-to-end delays can be seen in Figure 8. The deadline for the input-data messages is $\widetilde{d} = 40\,\mu$s, the release date for the output-data messages is $\widetilde{r} = 35\,\mu$s, according to the requirements shown in Section V. The end-to-end delays for the synchronisation message are set in such a way that the synchronisation message is forwarded without any delay in each node.

Within one communication cycle, it is ensured that all the input-data messages have been sent sooner than $40\,\mu$s after the start of the red interval (see Figure 5 and 6). Similarly, the output-data messages cannot be sent earlier than $35\,\mu$s after the start of the red interval. The synchronisation messages are still sent using the cut-through mechanism as in the previous figure. At first sight, the communication cycle is sparser and thus longer than in the case where no deadlines were posed. However, the time available for the controller application increased by $42.01\,\mu$s while prolonging the red interval only by $13.78\,\mu$s.

A. Computational Results

The algorithm presented in the previous sections has been tested on different Profinet IO IRT networks generated from

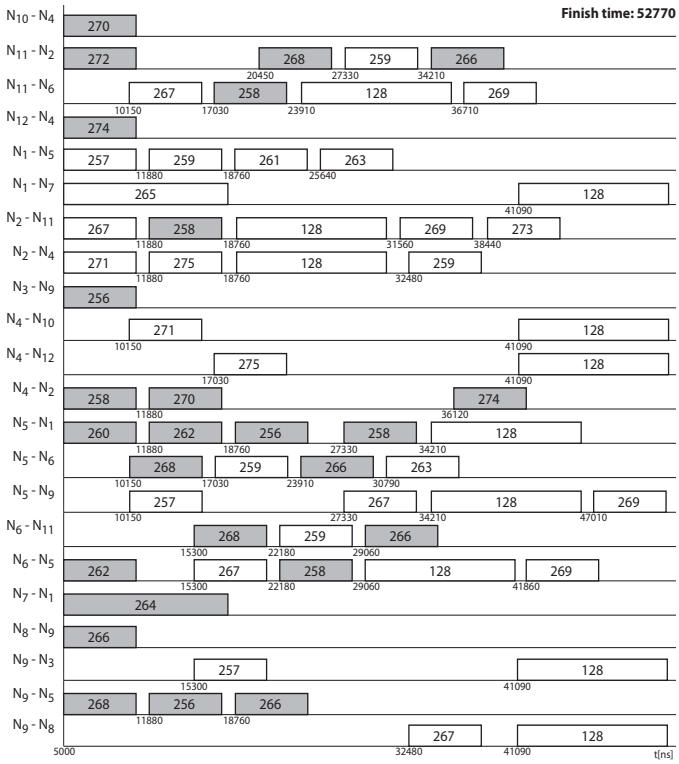

Fig. 7. Schedule without $\widetilde{d}$, $\widetilde{r}$, $\widetilde{e}$

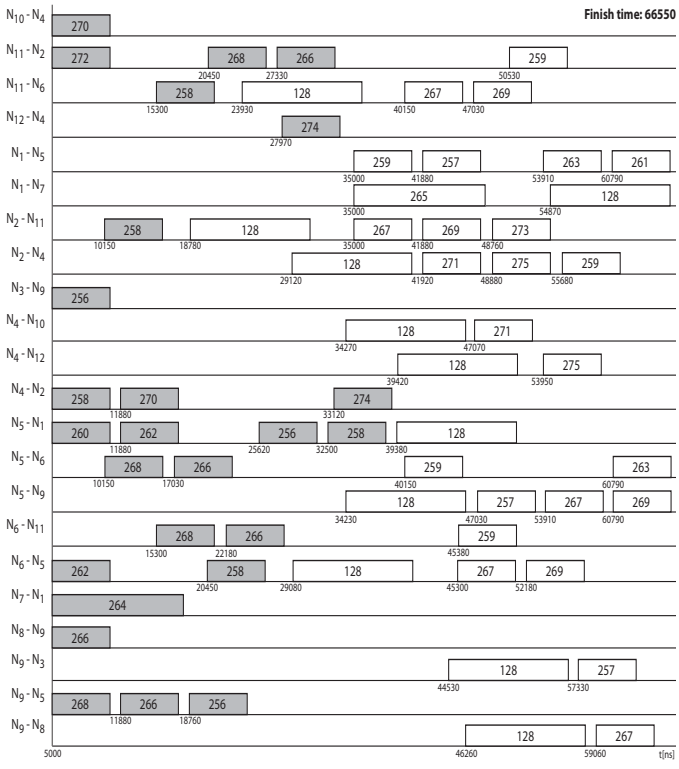

Fig. 8. Schedule with $\widetilde{d}$, $\widetilde{r}$, $\widetilde{e}$

the professional engineering tool [18]. All the networks contain Simotion D as the IO controller and Sinamics drives as the IO devices, and differ in the topology and the number of nodes. *Topology 1* resembles a star topology as there is an IO controller in the middle of a star, and 4 branches connected to its ports, each with 5 IO devices connected in a line. All other topologies represent a line and differ in the position of the IO controller (at the beginning, in the middle) and the number of the IO devices. In *topologies 3 and 4*, the IO controller is at the beginning and there are 20 and 30 IO devices, respectively. In *topologies 2, 5 and 6*, the IO controller is in the middle and there are 20, 30 and 40 IO devices, respectively. Table IV summarises the results, where n is the number of tasks, T_C is the computation time of schedule without the $\widetilde{d}$, $\widetilde{r}$, $\widetilde{e}$ constraints and C_{max} is the corresponding schedule length. $C_{max}^{Simatic}$ is the schedule computed by the professional engineering tool [18]. Thus, the values of C_{max} and $C_{max}^{Simatic}$ are directly comparable and their difference is very small. The positions of the tasks in the two schedules differ significantly, but $C_{max} \leq C_{max}^{Simatic} + 150$ns , which is a very small difference for the heuristic algorithm. The computation time of the Simatic schedule cannot be measured as it is part of the entire engineering tool and the schedule computation cannot be separated. Nevertheless, the computation of the Simatic schedule is faster because our tool is partly implemented in Matlab. The $C_{max}^{\widetilde{d},\,\widetilde{r},\,\widetilde{e}}$ values representing the schedule length with constraints, are influenced by posing $\widetilde{r}$, $\widetilde{d}$ and $\widetilde{e}$ on the messages and their computation time is in the same range as T_C.

Topology	n	T_C	C_{max}	$C_{max}^{Simatic}$
1	142	0.242 s	52.35 μs	52.2 μs
2	292	1.024 s	121.15 μs	121 μs
3	442	2.656 s	155.55 μs	155.4 μs
4	962	18.125 s	224.35 μs	224.2 μs
5	512	3.781 s	121.15 μs	121 μs
6	882	14.156 s	155.55 μs	155.4 μs

TABLE IV
COMPUTATION TIMES

VII. Conclusion and Future Work

The schedules with similar values of C_{max} can be produced by the algorithm presented here and by the one in the professional engineering tool [18] for a network with up to 40 devices and 1000 tasks. For smaller topologies, an optimal schedule was guaranteed because of using the Integer Linear Programming (ILP) algorithm, for large topologies a heuristic approach was used, which may differ to a small extent from the optimum schedule (as compared to the smaller topologies). Although the presented algorithm performs slower than the one of [18], it provides a flexible and fully documented solution.

The scheduling problem has been extended in order to separate the input and output messages for the controller

application to get more time for the control law computation or to ensure compact synchronisation message delivery throughout the whole network. This original approach offers great flexibility in specifying the application requirements which may by easily extended, for example by a synchronous delivery of the multicast messages.

In the current work we focus on the problem with different periods of IO data while using a superperiod concept. This problem fits in the formulation proposed in this paper, but we still have to evaluate its performance. Finally, we would like to focus on the adaptive behaviour of this formulation when new tasks are to be added to the existing schedule. Such a problem should be solvable by fixing the start times of the existing tasks in graph G and using the same optimisation algorithm. The time complexity in such a case should be related to the number of new tasks, thus allowing one to use optimal solvers. Another possibility is to use specific incremental algorithms working with graph G, based on the incremental Floyd or Bellman-Ford algorithms.

Acknowledgment: this work was supported by the Ministry of Industry and Trade of the Czech Republic under Project FT—TA3/044 and by the Ministry of Education of the Czech Republic under projects 2C06010 and 1M0567.

REFERENCES

[1] E. Tovar and F. Vasques, "Real-time fieldbus communications using profibus networks," *IEEE Transactions on Industrial Electronics*, vol. Vol. 46, No. 6, pp. pp. 1241–1251, 1999.

[2] E. Tovar, "Guaranteeing real-time message deadlines in profibus networks," in *Proceedings of the* 10^{th} *Euromicro Workshop on Real-time Systems*. IEEE Press, 1998.

[3] H. Kopetz, C. E. Salloum, B. Huber, R. Obermaisser, and C. Paukovits, "Composability in the time-triggered system-on-chip architecture," in *SoCC*, 2008, pp. 87–90.

[4] J. Sifakis, S. Tripakis, and S. Yovine, "Building models of real-time systems from application software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 100–111, 2003.

[5] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[6] E. T. Bjorn Andersson, Nuno Pereira, "Analysing tdma with slot skipping," *IEEE Transactions on Industrial Informatics*, vol. 4, pp. 225–236, 2008.

[7] J. Ferreira, L. Almeida, A. Fonseca, P. Pedreiras, E. Martins, G. Rodríguez-Navas, J. Rigo, and J. Proenza, "Combining operational flexibility and dependability in FTT-CAN," *IEEE Trans. Industrial Informatics*, vol. 2, no. 2, pp. 95–102, 2006.

[8] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real-Time Systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[9] M. Felser, "Real-time ethernet – industry prospective," *Proceedings of the IEEE*, vol. 93, 2005.

[10] J.-D. Decotignie, "A perspective on ethernet as a fieldbus," in 4^{th} *Int. Conference on Fieldbus Systems and Their Applications (FeT'01)*, 2001.

[11] P. Pedreiras and L. Almeida, "The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency," in *Proceedings of the* 14^{th} *Euromicro Conference on Real-Time Systems (ECRTS'02)*, 2002.

[12] S.-K. Kweon, K. G. Shin, and G. Workman, "Achieving real-time communication over ethernet with adaptive traffic smoothing," 6^{th} *IEEE Real Time Technology and Applications Symposium (RTAS'00)*, vol. 00, p. 90, 2000.

[13] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered Ethernet (TTE) design," 8^{th} *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, vol. 00, pp. 22–33, 2005.

[14] O. D. D. S. (DDSIG), "Data-distribution service for real-time systems (DDS)," http://portals.omg.org/dds.

[15] Profibus International, *Ethernet Powerlink V2.0, Communication Profile Specification*. Ethernet Powerlink Standardization Group, October 2007.

[16] ——, *Application Layer protocol for decentralized periphery and distributed automation, Specification for PROFINET, IEC 61158-6-10/FDIS*. Profibus International, October 2007.

[17] ——, *Application Layer services for decentralized periphery and distributed automation, Specification for PROFINET, IEC 61158-5-10/FDIS*. Profibus International, October 2007.

[18] Siemens, "Simatic manager," Technical documentation, 2008.

[19] U. Lauther, "An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background," *GI-Tage*, vol. 22, p. 219–230, 2004.

[20] F. Dopatka and R. Wismüller, "A top-down approach for real-time industrial-Ethernet networks using edge-coloring of conflict-multigraphs," in *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, 2006.

[21] IEEE, *802.1Q Standard for local and metropolitan area networks – virtual bridged local area networks – revision*. The Institute of Electrical and Electronics Engineers, Inc., New York, 2005.

[22] ——, *802.1D Standard for local and metropolitan area networks media access control (MAC) Bridges*. The Institute of Electrical and Electronics Engineers, Inc., New York, 2004.

[23] ——, *802.3 Standard: Part 3 – Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. The Institute of Electrical and Electronics Engineers, Inc., New York, 2005.

[24] R. Seifert, *The Switch Book*. Wiley Computer Publishing, 2000.

[25] M. Popp, *Industrielle Kommunikation mit PROFINET*. PROFIBUS Nutzerorganisation, 2007.

[26] J. Błazewicz, K. Ecker, G. Schmidt, and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, 2nd ed. Springer, 2001.

[27] B. Roy, "Graphes et ordonnancement," *Revue Française de Recherche Opérationnelle*, vol. 4, no. 25, pp. 323–333, 1962.

[28] P. Brucker, A. Drex, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3–41, 1999.

[29] C. Schwindt, *Resource Allocation in Project Management*. Springer, 2005.

[30] K. Neumann, C. Schwindt, and J. Zimmermann, *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2003.

[31] B. D. de Dinechin, "Simplex scheduling: More than lifetime-sensitive instruction scheduling," Proceedings of the International Conference on Parallel Architecture and Compiler Techniques, pp. 327–330, 1994.

[32] R. Alur and D. Dill, "A theory of timed automata," *J. Theoret. Comput. Sci.*, vol. 126, p. 183–235, 1994.

[33] P. Šůcha and Z. Hanzálek, "Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times," *Mathematical and Computer Modelling*, vol. 47, no. 9–10, pp. 925–942, May 2008, springer.

[34] W. Herroelen, B. D. Reyck, and E. Demeulemeester, "Resource-constrained project scheduling : A survey of recent developments," *Computers and operations research*, vol. 25, no. 4, pp. 279–302, 1998, elsevier.

[35] B. Franck, K. Neumann, and C. Schwindt, "Truncated branch-and-bound, schedule construction, and schedule-improvement procedures for resource-constrained project scheduling," *OR Spektrum*, vol. 23, pp. 297–324, 2001.

[36] Z. Hanzálek and P. Šůcha, "Time symmetry of project scheduling with time windows and take-give resources," 4th Multidisciplinary International Scheduling Conference: Theory and Applications. Dublin. August, 2009.