CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DIPLOMA THESIS

Scheduling and Visualization of Manufacturing Processes

Prague, 2008

Author: Roman Čapek

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Roman Čapek**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Rozvrhování a vizualizace výrobních procesů**

Pokyny pro vypracování:

1. Seznamte se s algoritmy pro rozvrhování výrobních procesů.
2. Navrhněte formát zadávání těchto problémů do nástroje TORSCHE.
3. Naprogramujte algoritmus pro rozvrhování na zdrojích s omezenou kapacitou založený na celočíselném lineárním programování.
4. Naprogramujte rozhraní pro vizualizaci těchto problému ve Virtual Reality Toolboxu pro Matlab.
5. Odzkoušejte a zdokumentujte vámi navržené řešení.

Seznam odborné literatury:

[1] M. Pinedo. Planning and Scheduling in Manufacturing and Services. Springer, USA, 2005.
[2] Y. Crama, V. Kats, J. van de Klundert, E . Levner. Cyclic scheduling in robotic flowshops. Annals of Operations Research, Volume 96, Numbers 1-4, November 2000

Vedoucí: Ing. Přemysl Šůcha, Ph.D.

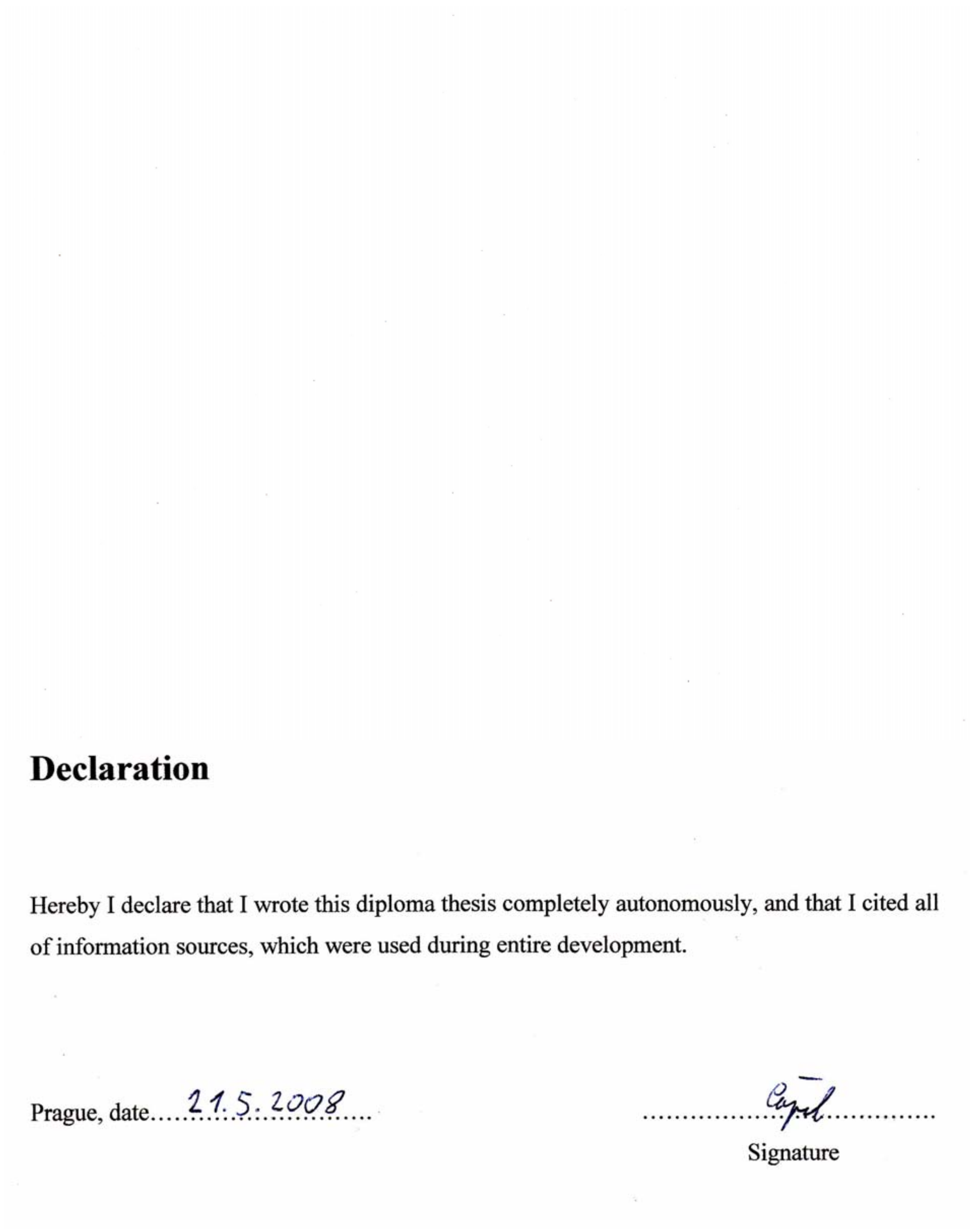
Platnost zadání: do konce zimního semestru 2008/2009

[illegible]
prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická -6- L.S.

v. z. [illegible]
doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 10. 9. 2007



Declaration

Hereby I declare that I wrote this diploma thesis completely autonomously, and that I cited all of information sources, which were used during entire development.

Prague, date 21.5.2008

[illegible]

Signature

Thanks

I would like to thank everbody who helped me either directly or indirectly during the work on this thesis. Especially to Ing. Přemysl Šůcha Ph.D., my supervisor, who has always given me advice willingly, sometimes even over the frame of his duties, and helped me with the work flow and the finalization of the thesis. Furthemore, I would like to thank to Doc. Dr. Ing. Zdeněk Hanzálek for his expert advices and also to my parents who support me during my entire study.

Annotation

This work involves results achieved in two parts of the scheduling area. A research in the scheduling with more alternative process plans is described in the first place. We suggest to represent this problem using the Petri nets formalism and we offer data structures and methods suitable for this problem. For this data representation, an algorithm based on Integer Linear Programming (ILP) is proposed and tested on randomly-generated data. Second part of this thesis is dedicated to description of utilization of the simulation and visualization in scheduling. Implementation of VISIS (VIsualization and SImulation in Scheduling), a tool for the simulation and visualization, is described and case studies and examples are presented.

Anotace

Tato diplomová práce je věnována popisu výsledků dosažených ve dvou částech teorie rozvrhování. Prvním řešeným problémem je rozvrhování s více alternativními výrobními plány. V práci je zahrnut výzkum navazující na již existující práce v oblasti alternativního rozvrhování a pro zadání struktury problému je využita notace Petriho sítí. Pro takto popsaný problém je navržen algoritmus založený na celočíselném lineárním programování. Navržené řešení je otestováno na náhodně generovaných instancích problému s alternativními výrobními plány. Druhá část této práce je potom věnována popisu využití vizualizace a simulace v oblasti rozvrhování. Práce rovněž obsahuje popis implementace nového nástroje pro vizualizaci a simulaci nazvaného VISIS (VIsualization and SImulation in Scheduling) a několik ukázkových příkladů vytvořených pomocí tohoto nástroje.

Content

List of Figures

Chapter 1

1. Introduction

1.1. The Visualization in Scheduling

Scheduling theory plays important role in optimization of resources that is used in many manufacturing and service industries. Many optimal and heuristic algorithms have been proposed for the scheduling area, but there is a growing demand for transparent and realistic representation of results in scheduling. The objective of the visualization is to bring these theoretical results near to non-experts in scheduling theory. Especially production scheduling and planning needs to be represented in the transparent form.

Visualization is a technique for creating images, diagrams or animations for graphical representation of real systems. It has expanding applications in science, education, engineering (e.g. production scheduling visualization), interactive multimedia, medicine, etc. Nowadays, the visualization is used for graphic representation of theoretical background in the first place. This representation needs to be as precise real world approximation as it is possible. Therefore, visualization should be designed for concrete instance of problem instead of representing similar problems by one predefined template. The best way to keep these demands is to bind established scheduling notation with user-defined virtual world definition.

Not only in relation with scheduling graphic representation can be used. Another area with wide use of the visualization is presentation of projects, their results and motivation examples. Transparent presentation of a final solution of any problem is very important. We can say almost as important as the solution itself. Also motivating students and their approach to new areas of study can be supported by the visualization.

We can consider three basic areas of the visualization use:

1) As a feedback for creating a time schedule.
2) For graphic representation of already scheduled problems, e.g. for production scheduling.
3) For demonstrational purposes, e.g. a final solution presentation.

1.2. Scheduling and the Simulation

The simulation is used in many contexts, including the modeling of natural systems or human systems in order to gain insight into their functioning. Other contexts include simulation of technology for performance optimization, safety engineering, testing, training and education. Simulation can be used to show the eventual real effects of alternative conditions and courses of action. Whereas visualization has main purpose in the presentation of scheduling results, simulation can serve as an optimization tool. We can use simulation not only in creating time schedules but also for optimizing system design, e.g. in digital processing. In both cases, simulation can be used for cyclic optimization via modification of some parameters by recalculation with acquired data or simply by estimation. In comparison with simulation using hardware, less time is needed for a programme realization of the same problem. Design of digital filters can be mentioned as a suitable example.

Easier modification of whole system is also one of the advantages of the computer simulation. In analog version, some parameters of hardware elements can be changed to modify function of the system. This change may not guarantee needed precision of new value; on the other hand, change of parameters in digital version is exactly defined and changes can be repeated infinitely. Consequently, we can say that computer simulation is less vulnerable to errors caused by inaccuracy of the project implementation.

Consequently, we can describe basic areas of computer simulation usage:

1) As an optimization tool for improving time schedules.
2) For determining the influence of given schedule to whole system function.
3) As an approximation of the real system and its function before hardware implementation.

Points 2) and 3) are very closely related to design of digital signal processing units, especially for digital filters. Usage of computer simulation there leads to high reduction of time needed to their design.

1.3. Optional Scheduling Motivation

Scheduling theory itself assumes exactly given set of tasks to be scheduled. This means that each given task is present in the final schedule, only start time and processor has to be assigned for each task. Nevertheless, there is at least one situation when classic conception of tasks is not sufficient: problem with *alternative process plans*. In related works, there are also used terms *alternative* (or *optional*) *tasks*, *activities* and *routings* [1], [2], [3], [4], [5]. In this work, it will be referred as the *optional scheduling problem* or as the *problem with alternative process plans*. Not only for production scheduling can this situation occur, but production scheduling is the most significant example from real world applications. Let us consider situation when there are more alternative routes to satisfy all demands and constraints and each route is formed by different set of tasks. Then not all of initial tasks will be present in the final schedule. We can say that presence of each task in the final solution is only optional, so the solving algorithm has to be able to choose and schedule only subset of tasks instead of creating schedule from all of them. Presence of precedence constraints between tasks results from the principle of the optional scheduling problem.

As mentioned above, this situation occurs mainly in production scheduling because there are usually more ways how to complete the product. For example, to repair a television one expert worker or two less skilled workers are needed. First alternative for chief of the workshop is to assign television repair to one skilled worker who needs about 2 hours to complete the work and second alternative is to commit it to two workers who need about 3 hours to do the same work. Each worker can use two different analyzer units where time to finish the repair also differs and these analyzer units can represent shared resources for all workers. All these facts lead to more than one way how to complete the set of demands under fixed constraints. Much more examples of this problem could be mentioned.

In fact, the optional scheduling problem is a combination of scheduling and planning areas. *Planning* consists of sequencing of operations to one integrated manufacturing plan. Only the operation itself is considered in the phase of creating the production plan independently on its real processing time and resource requirements. Precedence constraints are defined during the planning. *Scheduling* deals with tasks specifications like processing times, resource demands and precedence constraints. The goal of the scheduling phase is to create a time schedule optimizing some criterion while respecting all constraints.

1.4. Contribution

This diploma thesis presents results in two areas of scheduling theory [6]. One part contains description of the new tool for representation of scheduling results via the simulation and visualization. Second part is dedicated to an optional scheduling problem, its description, data representation and an optimal algorithm for solving of this problem is proposed.

1.4.1. VISIS Application

First goal of this diploma thesis is to present an application for demonstration of scheduling results in the Matlab environment (http://www.mathworks.com/) : *VISIS* (VIsualization and SImulation in Scheduling). This application uses the Matlab-based simulation environment Simulink and the Virtual Reality toolbox for the graphic visualization. Two areas of usage are considered: simulation for monitoring of the influence of the scheduling process on the system function (e.g. for digital filters) and time visualization (e.g. graphic representation of the execution on a production line in time). This tool will be freely available in the next version of TORSCHE Scheduling Toolbox for Matlab (http://rtime.felk.cvut.cz/scheduling-toolbox/) planed on October 2008. Up to our knowledge there is no such a tool providing visualization of scheduled processes in this range.

VISIS is able to simulate or visualize any time schedule given by TORSCHE data structures. The application is adjusted to maximize user comfort and simplicity of usage. Therefore, most of data and files needed for the simulation and visualization are created automatically and the structure of the code that defines activities is checked before the start of simulation in Simulink. Virtual reality world is fully user-defined so it can satisfy all appearance demands. In addition, parameters that can be changed via control code are defined by user of the application. Visualization can be realized in 2D or 3D world. Some examples were created as an illustration of the VISIS potential. All data history can be saved to later analysis and more, Virtual Reality toolbox used for visualization provides possibility to capture any frame or record runtime as a video file. Simulink allows stopping and restarting the simulation in any time and it is also possible to change some parameters during the simulation.

VISIS is designed to provide transparent representation of scheduling results in the first place but it can be also used for optimization purposes. Especially simulation using VISIS can serve as a fast feedback to acquired schedule and it can contemporaneously show function of whole simulated system. This can be great help for solving problems with assigning computing operations for multiprocessor systems and also for designing digital processing units at all.

1.4.2. Related Works for the Visualization and Simulation

This work comes up from TORSCHE Scheduling Toolbox for Matlab [7], which provides data structures and algorithms for time scheduling. Thus, all work is realized in the Matlab environment. One of the related tools is TrueTime [8] - a Matlab based tool for real-time simulation for wide spectrum of problems, e.g. digital filters, embedded systems or wireless networks. TrueTime is very strong simulation tool but utilization for small problems is quite difficult and especially for scheduling results simulation it is unnecessarily sophisticated solution. Fishman [9] made a comprehensive review of Discrete Event Simulation (DES) which is a system represented by sequence of disjunctive events. Each event occurs at an instant time and leads to change in the system state while there are not continuous states. Therefore, DES systems are very closely related to the simulation of scheduling results because each task is one discrete event, although it can contain more operations. But only the beginning and the end of each task is important for the simulation of a time schedule. Optimization using simulation is also described in this book. Another utilization of simulation-based optimization in real production was shortly described by Manlig and Sramek [10]. An alternative for purely sequential discrete event simulation was proposed by Misra [11]. This survey is dedicated to distributing of DES to more cooperating processors what may provide better performance of simulation. Utilization of simulation for production scheduling was discussed by Toal, Coffey and Smith [12], including also expert systems utilization.

For visualization purposes, OpenGL (Open Graphics Library) (http://www.opengl.org/) is a standard specification defining a cross-platform API for writing applications that produce 2D and 3D computer graphics. This means that visualization can be realized by OpenGL at any operation system. The visualization in scheduling was studied at Karlsruhe University [13] and some application for visualization of process scheduling has been developed there. This tool has predefined template where up to 8 processes (tasks) with some attributes and up to 4 processors can be defined. Time schedule is then represented by showing state of all tasks and processors in time. On the other hand, Matlab includes Virtual Reality toolbox, which is also sufficient tool for visualization of scheduling results. More tools can be mentioned but none of them provides visualization in close relations with user-defined time schedules.

1.4.3. Algorithm for the Optional Scheduling Problem

Second part of this thesis is dedicated to description of the new scheduling algorithm for the problem with alternative process plans. This involves not only realization of algorithm itself but also input data representation and a specification of the problem types that can be solved. Scheduling algorithm is based on Integer Linear Programming (ILP) [14] and is implemented in the Matlab environment using additional ILP solver. The formalism of Petri nets [15] in combination with TORSCHE toolbox structures and algorithms is used for input problem setting and solving.

Solution for a certain area of scheduling problems, including case with alternative process plans and minimizing C_{max} criterion in the first place, was implemented. Moreover, ILP algorithm is able to solve problems with given processing times, release times, deadlines and precedence constraints for tasks while one processor, dedicated processors or infinite amount of identical processors are available. We can say that each problem denoted by $\{1, P, PD\} \,|\, p_i, r_i, \tilde{d}_i \,|\, C_{max}$ [6] that can be interpreted via Petri nets is solvable. The proposed solution can be easily modified to optimizing another criterion that can be described as a linear combination of tasks start times and processing times.

Definition of the problem structure using PN notation allows easy modification of the created project via simple graphic interface. Output of the application is the standard TORSCHE toolbox structure *taskset* with the included time schedule that can be depicted as a Gantt chart [6]. It is also possible to use VISIS application to visualize or simulate acquired solution.

1.4.4. Related Works for the Optional Scheduling Problem

There were some attempts to deal with the optional scheduling problem but no one in the same form as described in this work. Bartak [1] proposed edge-finding algorithm for scheduling with optional activities (tasks) with unary resource. This algorithm is able to solve the problem with given processing time, earliest possible start time (release time) and latest possible completion time (deadline) for each task. The function of the algorithm is based on tightening the time bounds for tasks and their progressive elimination from the initial set of tasks. Beck and Fox [2] formulated a constraint-based representation of optional activities to model problems containing alternatives in scheduling. They begin with listing of all possible scheduling alternatives and then they connect these different ways into one temporal graph with so-called XOR nodes. Each of the XOR nodes represents one branching to more alternatives where decision has to be made. Beck and Fox assign each task with probability of existence (PEX) value, bounded within interval $\langle 0, 1 \rangle$. Rules for propagation of these values through the graph are then defined and rules for connection of graph nodes are also described. Exact working algorithms and some heuristics are proposed for solution of problems set by this graph. This representation by the graph with established XOR nodes is in very close relation with Petri nets formalism used in this diploma thesis. Wilhelm [16] proposed column model represented by state transition graph to deal with assembly system design problem with tool changes. Each node in the graph represents one station with single operation assigned and determined time needed for this operation. Algorithm for solution of this problem is then based on finding the shortest path through the graph respecting given constraints. Helimann [17] established model for project scheduling with multi-mode resources where each activity can be realized on more than one resource while processing times and also minimum and maximum time lags for other tasks differ. Solution is set on branch and bound method with depth-first search.

Utilization of Petri nets in scheduling area was described by Tuncel and Bayhan in [18]. They discuss application of Petri nets in production scheduling and then they describe combination of PN formalism with search algorithms, heuristics, meta-heuristics and mathematical based algorithms like linear programming for minimizing the cycle time of the system. Algorithm for minimizing total tardiness in a flexible manufacturing system based on Petri nets formalism was proposed by Mejia and Montoya [19]. They use transitions to represent events (tasks) and places to represent states, conditions and machines (processors). This approach is very close to the problem representation described in this thesis except the processors representation. Solution of the problem is based on state equations of the system and a heuristic search algorithm is also described.

1.5. Paper Organization

The paper is divided into eight chapters. First chapter presents motivation of this thesis and its contribution in two areas: simulation and visualization of scheduling results in the first place and optional scheduling as second. Related works are also described there. Chapter 2 provides basic scheduling theory description, then some examples of scheduling problems are described and representation of their results is discussed. Statement of the optional scheduling problem is provided at the end of this section. Next chapter is dedicated to description of the optional scheduling problem representation and its relation to Petri nets formalism. Chapter 4 describes Integer Linear Programming formulation for the optional scheduling problem. Chapter 5 describes areas of utilization of both visualization and simulation tools. Following chapter is dedicated to description of implementation of VISIS application in Matlab, its relation to TORSCHE toolbox and short description of created functions and other files. Chapter 7 presents experiments, case studies and examples for both parts of this diploma thesis. Last chapter contains conclusions of this work, contribution to scheduling area and future utilization of VISIS application and the optional scheduling algorithm.

Chapter 2

2. Problem Description

2.1. Scheduling Theory

Scheduling is an optimization of resources usage with given constraints in time. In other words, scheduling solves the problem how to assign given resources to given tasks in time. Scheduling problems are characterized by three basic sets [6]:

1) Set of tasks - $\mathcal{T} = \{T_1, \dots, T_n\}$
2) Set of processors (machines) - $P = \{P_1, \dots, P_m\}$
3) Set of additional resources - $R = \{R_1, \dots, R_s\}$

Tasks are determined by several numerical parameters, e.g. processing time. There are two general constraints: each task can be processed by at most one processor at a time and each processor is able to process at most one task at a time. Set of processors *P* determines amount and type of processors that can be used. Resources needed for execution of tasks, which are not included in set *P*, are represented by the set of additional resources *R*. For deterministic problems, $\alpha|\beta|\gamma$ notation was established [6], [20]. The first field α describes available resources (processors), β represents tasks and resources characteristics and γ denotes an optimality criterion. Detailed description is below. Many algorithms were proposed for solution of problems described by this notation.

Generally, scheduling is NP-hard problem. Polynomial algorithms are known only for the limited amount of scheduling problems, especially for problems with only one processor. For the rest of the problems, solving time needed to find optimal solution is exponentially proportional to size of the input problem. Algorithms can be based on searching techniques like branch and bound method, on constraint programming [21] or on integer linear programming [14]. Other way is to use some polynomial heuristic method but without assurance of optimal solution.

Production scheduling is a branch of scheduling mostly aimed to automated production lines and industrial production at all [22]. Almost all cases of production scheduling are NP-hard problems.

2.1.1. Definition of the Fundamental Scheduling Objects

There are some essential notions from scheduling theory used in this work:

Task - Basic scheduling object, referred to as T_i.

- Represents one concrete real world operation.

- Each task has its own execution time called processing time p_i.

- May have some additional properties:

- Release time r_i, which is the time at which T_i is ready for processing.
- Due date d_i, which is the time limit by which T_i should be completed.
- Deadline $\tilde{d}_i$, which is a time limit by which T_i must be completed.
- Weight w_i, which represents relative urgency of task T_i.
- Resource specification: on which resource has to be T_i processed.

- Scheduled task has its own start time s_i and completion time c_i in the schedule.

All parameters p_i, r_i, d_i, $\tilde{d}_i$ and *wi* are supposed to be positive integers.

Taskset - Set of tasks T_i, referred to as $\mathcal{T}$.

- All tasks have the same resource environment.

Job - Set of tasks T_i, referred to as *J*.

- Each task has its own resource specification - dedicated processors problem.

Problem - Description of tasks, resources and constraints.

- Established $\alpha|\beta|\gamma$ notation was described by Blazewicz [6].

- α characterizes the type and amount of processors used:

Ø	- one processor.
P	- identical processors.
Q	- uniform processors.
R	- unrelated processors.
PD	- dedicated processors, definition proposed by Kellerer and Strusevich [23].
O	- dedicated processors, open shop system.
F	- dedicated processors, flow shop system.

J - dedicated processors, jobshop system.

k - number of processors; integer value, written after type.

- β describes tasks and resources characteristics:

  pmtn - preemption allowed.

  prec - precedence constraints between tasks exists.

  p_i, r_i, d_i, $\tilde{d}_i$ - detailed description of processing times, release times, due dates and deadlines.

- γ denotes the optimality criterion; some of the most used criterions:

  C_{max} - minimizing of the latest completion time. Represents throughput of the system.

  $\sum C_i \cdot w_i$ - minimizing of the weighted sum of completion times.

  $\sum U_i$ - minimizing of the number of delayed tasks.

  L_{max} - minimizing of the largest L_i value, where $L_i = c_i - d_i$.

Schedule - Result of the scheduling.
- Describes allocation of tasks to resources in time.
- Mostly represented by Gantt chart [6] with time on *x*-axis and resource on *y*-axis.

2.2. Examples of Scheduling Problems

In this subsection, some examples of the scheduling problems are shown.

$\mathbf{1}|\mathbf{r_i}, \mathbf{\tilde{d}_i}|\mathbf{C_{max}}$ - problem with one processor, given processing times, release times and deadlines for tasks. Optimality criterion is to minimize the latest completion time. Let us have a set of four tasks, determined by the vector of processing times $p = [2,1,2,2]$, the vector of release times $r = [4,1,1,0]$ and the vector of deadlines $\tilde{d} = [7,5,6,4]$. Graphical representation of this problem via modified Gantt chart is shown in Figure 2.1. This picture was acquired using TORSCHE toolbox. Each task has its own processing time (width of appropriate rectangle area), release time (arrow aiming up) and deadline (arrow aiming down).

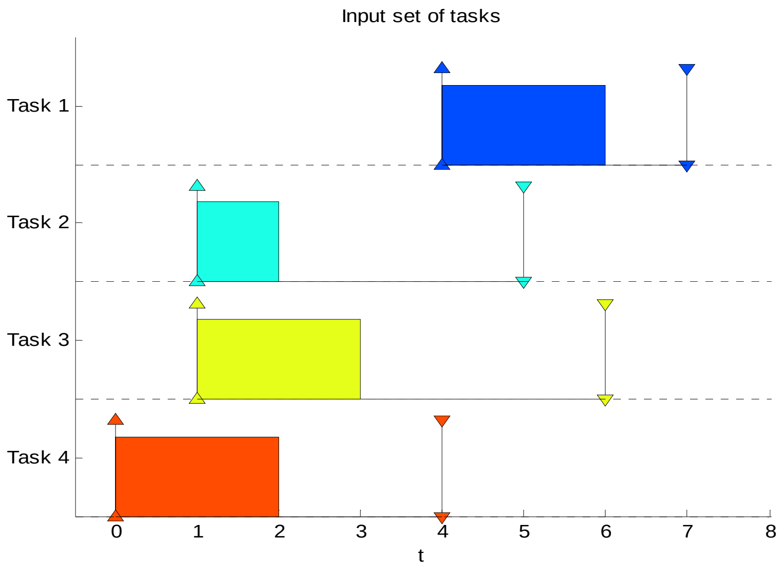


Figure 2.1 - $1|r_i, \tilde{d}_i|C_{max}$ problem

$\mathbf{P2|prec|L_{max}}$ - problem with two identical parallel processors, given processing times, due dates and defined precedence constraints between tasks. Presence of due date values is not involved in problem definition, but it is a logical consequence of the criterion L_{max}, which is subject to due dates. Optimality criterion is to minimize the largest L_i, where $L_i = c_i - d_i$. Value L_{max} can be also negative - in the situation when all tasks are completed before their due dates. Precedence constraints can be defined by the task-on-node graph [6], shown in Figure 2.2. Each node of this graph represents one task and each edge denotes a precedence constraint between appropriate tasks.

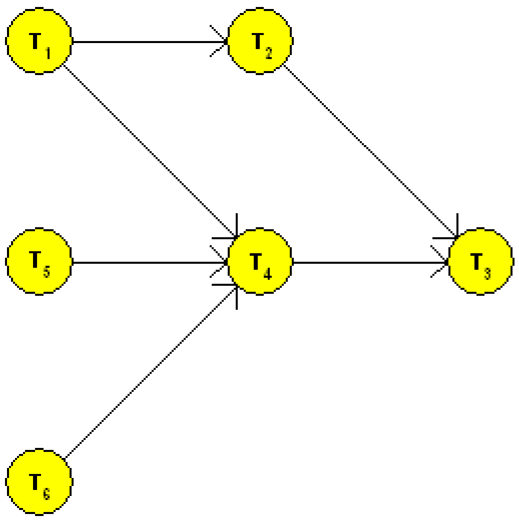


Figure 2.2 - Precedence constraints

2.2.1. Gantt Chart

As mentioned above, the most used representation of time schedules is Gantt chart. All task parameters and precedence constraints can be displayed in one chart. There are discrete time values on *x*-axis and description of processors on *y*-axis. Each start time of the task is represented by shift of appropriate task position along *x*-axis and assignment of the task to processor is determined by position on *y*-axis. Let us take an example of $1|r_i, \tilde{d}_i|C_{\max}$ problem from the previous subsection. Gantt chart resulting from the scheduling process, for example using branch and bound method, is shown in Figure 2.3.

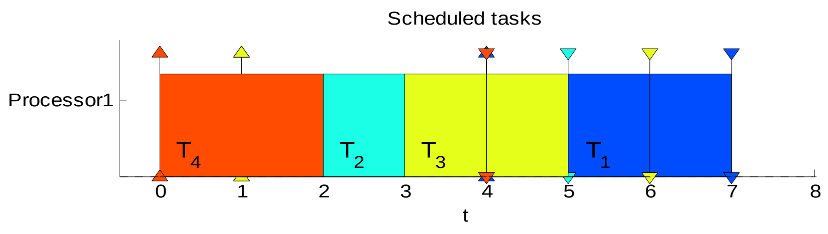


Figure 2.3 - Scheduled problem $1|r_i, \tilde{d}_i|C_{\max}$

Problem $P2|prec|L_{\max}$ from the previous subsection is taken as an example of multi-processor Gantt chart. Taskset characteristics: set of processing times $p = [2,2,2,3,3,3]$ and set of due dates $d = [3,5,8,6,4,3]$. Resulting schedule with precedence constraints taken from Figure 2.2 is displayed in Figure 2.4.

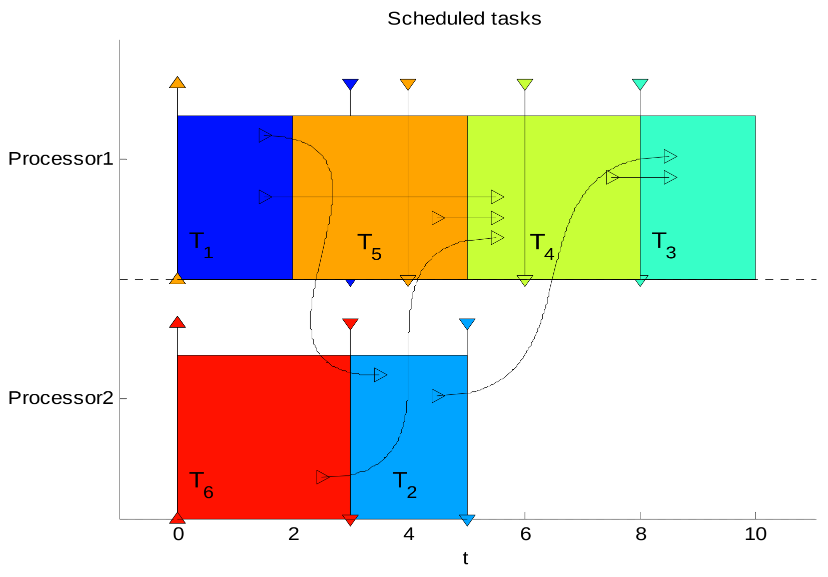


Figure 2.4 - Scheduled problem $P2|prec|L_{\max}$

2.2.2. The Hoist Scheduling Problem

The hoist scheduling problem [24], [25] deals with the problem how to schedule the hoist moves to perform the material handling tasks in the system. The most frequent used optimality criterion is C_{max} value because it represents throughput of the system. In the hoist scheduling problem, each task represents one move of the hoist with the material. The material has to be processed in several tanks with liquid and the time needed for processing in every tank is determined by its minimum and maximum value. In addition, all empty hoist moves have to be taken into account for the scheduling. Result of scheduling depicted by Gantt chart is shown in Figure 2.5 and representation by a special chart for the hoist scheduling problem is shown in Figure 2.6.

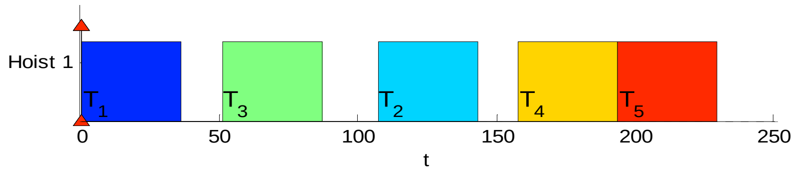


Figure 2.5 - Hoist scheduling Gantt chart

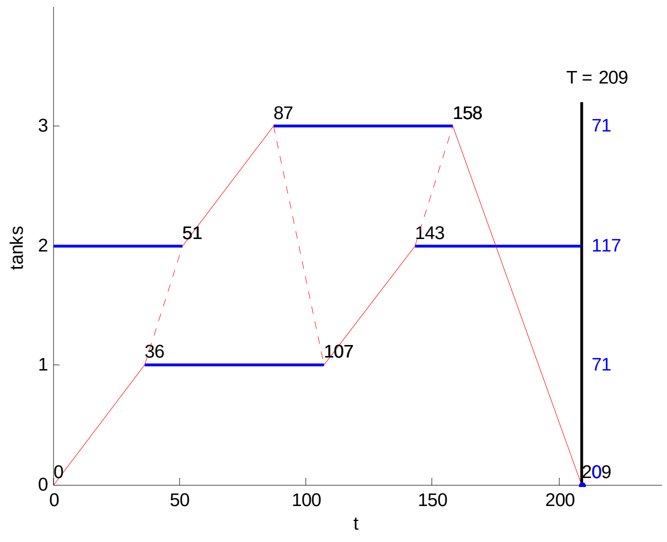


Figure 2.6 - Hoist scheduling special chart

Special chart for the hoist scheduling problem gives better idea of the acquired result, although understanding is quite difficult for the first time. Red solid lines in Figure 2.6 represent moves of the hoist with the material, red dashed lines depict empty hoist moves and blue lines represent temporary stays of the material in tanks. Labels of tanks are on *y-axis* and discrete time is on *x-axis*.

2.3. Optional Scheduling Problem Statement

In contrast to classic conception of tasks described in the previous section, optional scheduling problem is a problem that involves different approach for solving. More possible process plans are defined and only one of them has to be chosen. *Process plan* is a sequence of tasks that satisfies all input demands. All tasks can be included in more than one plan, but each sequence is determined by a disjunctive set of tasks, i.e. no task can be included in one process plan more than once. We can say that each process plan represents one alternative routing to complete the input assignment. Not all given tasks will be then present in the final schedule, so they can be called *optional tasks* [1]. These tasks have the same properties as described in Section 2.1 and also processors have the same characteristics. The only difference is that only one process plan has to be chosen and scheduled, so only one particular sequence of tasks is taken into account for scheduling.

This is very close to combination of planning and scheduling areas, but all decisions are made only on the basis of the scheduling algorithm. No additional information except the definition of tasks and alternative process plans has to be set before the scheduling process. The goal of the scheduling for the optional scheduling problem is to choose one process plan and assign all included tasks to processors according to the given criterion and respecting all given constraints. Result is then classic time schedule containing selected tasks.

Algorithm for the optional scheduling problem has to be different from classic scheduling algorithms from the point of view that not all constraints resulting from the problem assignment have to be satisfied. For example, if one task is included in two different process plans, then only precedence constraints from the chosen plan are taken into account for the scheduling process. Another problem is with task start time in the schedule: if all plans would be scheduled, the same task in different plans could have different start time. This is in contradiction with the fact, that each task can have only one start time for deterministic problems.

Chapter 3

3. Optional Scheduling Problem Representation

In order to develop a new algorithm for the optional scheduling problem, the input data representation has to be defined first of all. Two general demands have to be satisfied. Proposed data representation must be able to describe more alternative process plans in one instance of a problem and data entry should be comprehensible and should allow easy modification. For this purpose, existing related work is used as the starting point and the proposed solution is then modified using the Petri nets formalism.

3.1. Modified Temporal Network - XOR Graph

As mentioned in the previous section, the optional scheduling problem involves approach different from the situation when all given tasks are present in the final schedule. For this purpose, we will come out of the model proposed by Beck and Fox [2] and then this model will be modified and interconnected with the Petri nets formalism. Beck and Fox start with listing of all alternative process plans as a disjunctive set of task consequences, shown in Figure 3.1 (taken over from [2]).

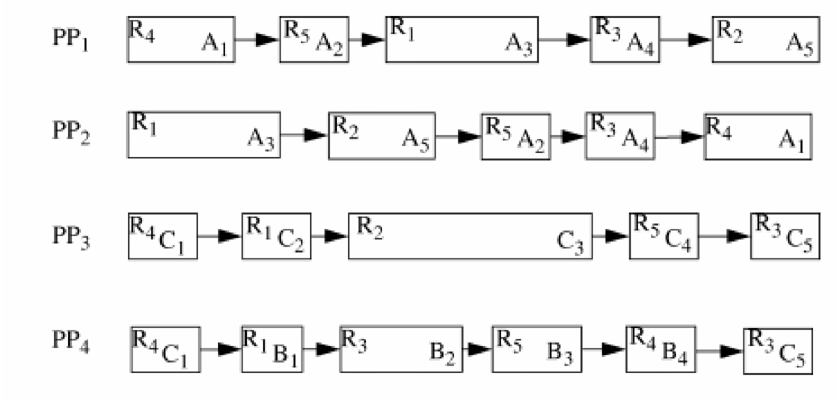


Figure 3.1 - Alternative process plans

Each task has its own identifier, written in the lower-right corner of the box, and resource to be processed on, written in the upper-left corner of the box. Each process plan (PP) is determined by a consequence of disjunctive tasks. The goal of the scheduling process is then to choose one of those process plans and create a time schedule of included tasks respecting their characteristics, resources specification and precedence constraints. As we can see in Figure 3.1, each task can be included in more than one process plan, but each process plan is determined by a disjunctive set of tasks. Task presence in the final solution is determined by a *probability of existence* - *PEX* value. As a next step, all process plans are connected into one graph - *modified temporal network*, shown in Figure 3.2. This graph, also called *XOR graph*, consists of three types of nodes:

1) *Activity node* is representation of task. Start time and completion time of an Activity node in the final schedule are temporal variables. Duration of an activity node in the temporal graph is determined by the processing time of an appropriate task.

2) *AND node* has start time and completion time in the schedule and these values are temporal variables. Duration of AND node is zero. For the PEX value propagation, two rules are defined: 1) all graph nodes linked to an AND node exist in the solution if and only if the AND node does; 2) All non-XOR nodes directly connected to an AND node must have the same PEX value as the AND node itself.

3) *XOR node* represents the possibility of choice during the scheduling process. Duration of a XOR node is zero and its start time and completion time are temporal variables again. Rules for PEX propagation through the XOR nodes are these: 1) At most one node connected to a XOR node upstream and one connected downstream can be present in the final schedule; 2) If there is a node connected to the XOR node downstream (upstream), which is present in the final schedule, then there must be just one node connected to the XOR node upstream (downstream) that is present in the final schedule and also XOR node itself must be present in the schedule.

Figure 3.2 (taken over from [2]) displays XOR graph resulting from the problem assignment shown in Figure 3.1 where four different process plans are defined. All these plans are interconnected by one XOR node at the beginning and one at the end. Moreover, plans 3 and 4 have the same first activity (task), so we can use branching via XOR node up to this activity. Similarly, the last activities of plans 3 and 4 are also the same, so we can connect these plans via XOR node before this activity node.

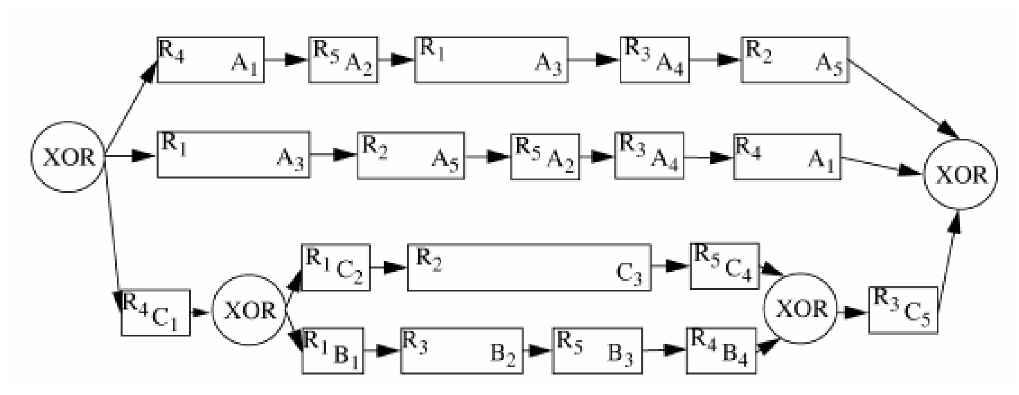


Figure 3.2 - XOR graph

3.2. Petri Nets in Optional Scheduling

In the following text, we modify XOR graph presented in the previous section and then we use the Petri nets formalism [15] to set the input problem. For this purpose, the Petri nets formalism has to be defined in the first place.

3.2.1. Petri Nets Basics

A Petri net is a formalism for modeling of systems with discrete states and events. Petri net (PN) is a quintuple $\{P, \mathcal{T}, F, W, M_0\}$, where $P = \{P_1, \ldots, P_m\}$ is the finite set of places, $\mathcal{T} = \{T_1, \ldots, T_n\}$ is the finite set of transitions, $F \subseteq (P \times \mathcal{T}) \bigcup (\mathcal{T} \times P)$ is the set of arcs, $W : F \to \{1, 2, \ldots\}$ is a weight function, $M_0 : P \to \{0, 1, 2, \ldots\}$ is the initial marking [15]. P and $\mathcal{T}$ are disjunctive, i.e. no object can be both a place and a transition. Petri nets are represented by a bipartite directed graph. Every arc of the graph connects one place and one transition; arc cannot connect two nodes of the same type. Consequently, places and transitions are regularly alternating in a graph. Transition without input place is called *source transition* and transition without output places is called *sink transition*. The same situation occurs for places. For the purpose of modeling and simulating of systems, each place can contain any number of *tokens*. A distribution of tokens over the places of a Petri net is called *marking M*. An example of a simple Petri net containing three places and three transitions is depicted in Figure 3.3. Marking of this net is $M = (1, 0, 0)$.

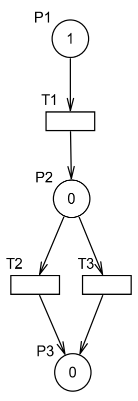


Figure 3.3 - Example of Petri net

The most important parameters of a Petri net for scheduling are state-transitions matrices W^+ and W^- and resulting *incidence matrix W*. W^- is defined as $m \times n$ matrix, where *m* is amount of places and *n* is amount of transitions. Columns of matrix W^- represent transitions and rows represent places. Each element of a matrix determines number of tokens taken from a place by an appropriate transition. Similarly, W^+ is defined as $m \times n$ matrix where each element of a matrix determines number of tokens added to a place by an appropriate transition. Incidence matrix of a Petri net is then defined as $W = W^- - W^+$. Generally, arcs between places and transitions can have different weights but for the optional scheduling problem purposes, all arcs are supposed to have weight equal to one. Also marking is ignored in that case.

State-transition matrices resulting from the Petri net depicted in Figure 3.3:

$$W^- = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad W^+ = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Incidence matrix:

$$W = W^+ - W^- = \begin{pmatrix} -1 & 0 & 0 \\ 1 & -1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$$

There are some special PN structures that are important for the optional scheduling problem. Possibility of choice, needed for the data setting, is represented by a place with more output transitions. Similarly, merging of more alternative process plans is modeled as a place with more input transitions. Only one of tasks T_1, T_2 and T_3 can be present in the solution of problem depicted in Figure 3.4.

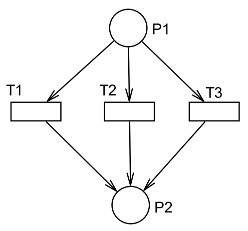


Figure 3.4 - Choice representation in PN

Situation with a process plan containing tasks that can be executed simultaneously is represented by a transition with more output (and then input) places (see Figure 3.5). Both tasks T_2 and T_3 have to be scheduled and they can be executed in the same time if there is sufficient resource environment.

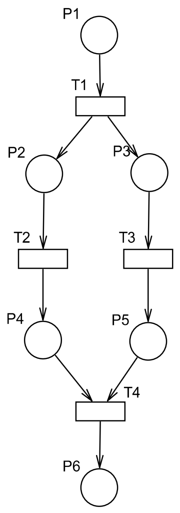


Figure 3.5 - Parallel process plans in PN

Not all models that can be defined via Petri nets are correct representations of the real process plans. There are some cases inconsistent with the idea of the process plans; one incorrect instance of Petri net is shown in Figure 3.6. The problem is in the fact that there is a branching into two alternatives and then these disjunctive parts are connected by one transition. This is classic example of deadlock because in the beginning, only one of two alternatives has to be chosen and later, both process plans need to be executed to reach the final state.

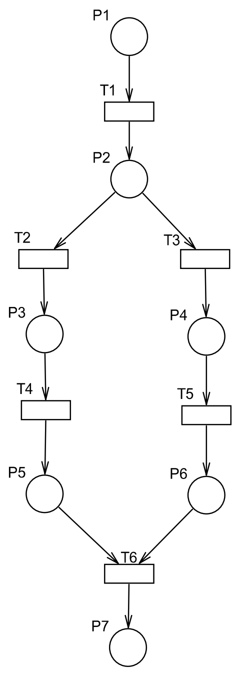


Figure 3.6 - Incorrect PN

3.2.2. Conversion of XOR Graph to Petri Net

In our case, transitions stand for activities (tasks) and places stand for the states of the system. In comparison with XOR graph, transitions represent both AND nodes and Activity nodes and places represent XOR nodes. This approach respects all rules for nodes in XOR graph defined in the previous text. All places connected with a transition are present in the final solution if and only if the transition is also present. For each place that is present in the final solution, there is just one input transition and one output transition, except the place representing initial state of the system (no input transitions) and place representing the final state of the system (no output transitions). With this presumption, we can modify the optional scheduling problem depicted in Figure 3.1 and Figure 3.2 to the version based on the Petri nets formalism (see Figure 3.7).

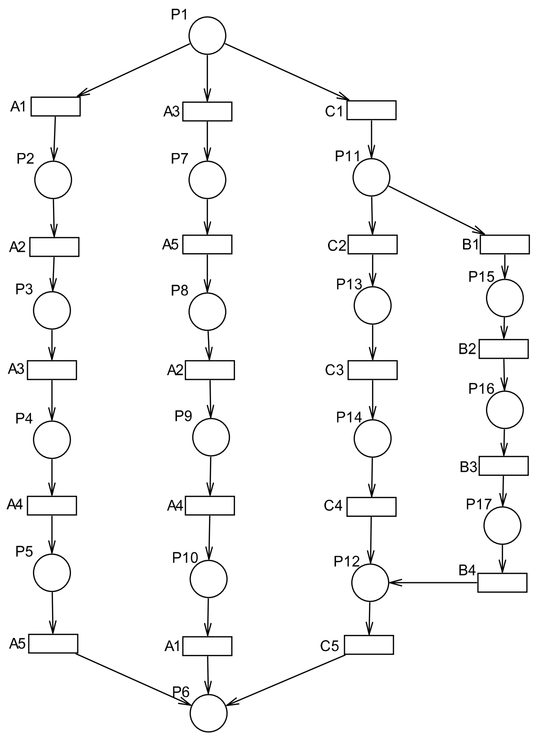


Figure 3.7 - PN for the optional scheduling problem

As mentioned above, possibility of choice during scheduling is modeled by a place with more output (and later input) transitions. At the beginning, only one of the tasks A_1, A_3 and C_1 has to be chosen and appropriate process plan has to be scheduled then. In the situation when C_1 is chosen, another one decision has to be made.

Input representation of the optional scheduling problem is defined as follows:

1) Each task is represented by a transition in a given Petri net.
2) Alternative process plans are modeled by a place with more output (and later input) transitions.
3) Parallel tasks are modeled by a transition with more output (and later input) places.
4) Petri net for the optional scheduling problem must have one source place, representing the start point for scheduling, and one sink place, representing the requested final state.
5) No source or sink transitions are allowed.

Any PN editor that provides possibility to obtain the state-transitions matrices W^+ and W^- can be used for the input problem setting. Description of the Petri net by the state-transition matrices is passed to the proposed scheduling algorithm. Tasks are determined by their processing times, release times, deadlines and resource specification if needed. All these parameters are passed to the algorithm as well.

Chapter 4

4. Algorithm for the Optional Scheduling Problem

In this section, an optimal algorithm based on Integer Linear Programming (ILP) for the optional scheduling problem is proposed. We use the Petri nets formalism described in the previous section to define the structure of the alternative process plans. State-transitions matrices W^+ and W^- are passed as input data to the algorithm. GLPK (GNU Linear Programming Kit) solver [26] is used to solve the ILP problem. GLPK package is intended for solving large-scale linear programming (LP) and mixed integer programming (MIP). It is a set of routines written in ANSI C and organized in the form of a callable library. This tool is freely available at http://www.gnu.org/software/glpk/.

4.1. Definition of Parameters and Variables

The goal of the algorithm is to chose and then schedule a subset of the initial set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$, represented by identical set of transitions. Each task is determined by its processing time p_i and the structure of the problem is defined via Petri net. Deadline $\tilde{d}_i$, release time r_i and dedicated processor number R_i can be also defined. Position of a task in the schedule is determined by its start time s_i. Presence of a task T_i in the final schedule is determined by a binary decision variable v_i where $v_i = 1$ if T_i is present in the final solution and $v_i = 0$ otherwise. For the purpose of ILP formulation for the optional scheduling problem, a binary decision variable $e_{i,j}$ is defined as presence of arc between place T_i and transition P_j in the final solution, $e_{i,j} = 1$ if an appropriate arc is present in the final solution and $e_{i,j} = 0$ otherwise. Further, let $x_{i,j}$ be a binary decision variable such that $x_{i,j} = 1$ if and only if $s_i + p_i \le s_j$ (i.e. T_i is followed by T_j) and $x_{i,j} = 0$ if and only if $s_i \ge s_j + p_j$ (i.e. T_j is followed by T_i) [27]. Objective of the algorithm is to minimize C_{max} defined as $C_{\max} = \max_i (s_i + p_i)$.

From a defined Petri net, only state-transitions matrices W^+ and W^- are taken into account for scheduling. Each element of the resulting matrix *W* determines number of tokens removed or added to a place (index of a row) by an appropriate transition (index of a column). In our case, all elements are from set $\{-1, 0, 1\}$ because all arcs in a given Petri net must have value equal to one. $W[i, j] = 1$ denotes that one token is added to place P_i by transition T_j, $W[i, j] = -1$ denotes that one token is removed from place P_i by transition T_j and $W[i, j] = 0$ means that there is no arc between place P_i and transition T_j. Another parameter, *placeType*, is computed from the matrix *W*. This parameter is a set with the same size as the set of PN places *P* and determines the type of each place; $placeType_i = 1$ denotes that place P_i is a sink place, $placeType_i = -1$ signifies that place P_i is a source place and $placeType_i = 0$ denotes that place P_i has both input and output transitions.

List of all parameters and variables used in the ILP formulation is summarized below.

Parameters:

n - number of tasks; also number of transitions in a Petri net.

m - number of places in a Petri net.

UB - sufficiently high positive integer constant.

$\mathfrak{T} = \{T_1, \ldots, T_n\}$ - set of tasks; equal to set of PN transitions.

$P = \{P_1, \ldots, P_m\}$ - set of PN places.

$W = W^- - W^+$ - incidence matrix of a Petri net; size of *W* is $n \times m$.

$R = [R_1, \ldots, R_n]$ - numbers of processors for dedicated processors problem.

$p = [p_1, \ldots, p_n]$ - vector of processing times.

$r = [r_1, \ldots, r_n]$ - vector of release times.

$\tilde{d} = [\tilde{d}_1, \ldots, \tilde{d}_n]$ - vector of deadlines.

$placeType = [placeType_1, \ldots, placeType_m]$ - indication of the source and sink places of a Petri net.

Variables:

si - start time of task T_i, $i = 1, 2, \ldots, n$; integer variable.

vi - presence of task T_i in the final schedule; $i = 1, 2, \ldots, n$; binary variable.

ei,j - presence of arc between place P_i and transition T_j in the final schedule; $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$; binary variable.

xi,j - allocation of tasks T_i and T_j in the time; $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$; binary variable.

- $x_{i,j} = 1$ means that T_i precedes T_j.
- $x_{i,j} = 0$ means that T_j precedes T_i.

Cmax - Value of the criterion; latest completion time.

4.2. Integer Linear Programming Formulation

Let us describe basic idea of the proposed ILP formulation for the optional scheduling problem. Three similar ILP models are described, one for the problem with one processor, one for infinite amount of parallel processors and the last one for the problem with dedicated processors. The goal of the ILP formulation is to chose and then schedule a subset of the set of tasks $\mathcal{T}$ subject to C_{max} criterion.

4.2.1. One Processor

ILP model for the problem with one processor is described in the first place:

$$s_i + (1 - v_i) \cdot UB \quad \geq \quad r_i \qquad i = 1, 2, \ldots, n \qquad (1)$$

$$s_i \quad \leq \quad \widetilde{d}_i - p_i + (1 - v_i) \cdot UB \qquad i = 1, 2, \ldots, n \qquad (2)$$

$$p_j \quad \leq \quad s_k - s_j + UB \cdot (2 - v_j - v_k)$$
$$i = 1, 2, \ldots, m; j,k = 1, 2, \ldots, n; W[i,j] > 0 \text{ and } W[i,k] < 0 \qquad (3)$$

$$p_j \quad \leq \quad s_i - s_j + UB \cdot x_{i,j} + UB \cdot (2 - v_i - v_j)$$
$$i,j = 1, 2, \ldots, n; k = 1, 2, \ldots, m; i < j \text{ and } W[k,i] > 0 \text{ and } W[k,i] \neq \pm W[k,j] \qquad (4)$$

$$s_i - s_j + UB \cdot x_{i,j} \leq UB - p_i + UB \cdot (2 - v_i - v_j)$$
$$i, j = 1, 2, \ldots, n; k = 1, 2, \ldots, m; i < j \text{ and } W[k,i] > 0 \text{ and } W[k,i] \neq \pm W[k,j] \qquad (5)$$

$$\sum_{j=1,2\ldots,n\,:\,W[i,j]<0} (e_{i,j}) = \sum_{k=1,2\ldots,n\,:\,W[i,k]>0} (e_{i,k}) - placeType_i \qquad i = 1, 2, \ldots, m \qquad (6)$$

$$e_{k,i} = e_{j,i} \qquad i = 1, 2, \ldots, n; j, k = 1, 2, \ldots, m; W[k,i] > 0 \text{ and } W[j,i] < 0 \qquad (7)$$

$$v_i = e_{j,i} \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, m; W[j,i] \neq 0 \qquad (8)$$

$$s_i + p_i \leq C_{max} + UB \cdot (1 - v_i) \qquad i = 1, 2, \ldots, n \qquad (9)$$

We start with two equations (1) and (2) representing the constraints resulting from release times and deadlines of tasks. If task T_i is not present in the final schedule ($v_i = 0$) then the constraints (1) and (2) are satisfied due to high positive constant *UB* and value of s_i is arbitrary. If $v_i = 1$ then the task cannot start before its release time and must be completed before its deadline. Precedence constraints between tasks are taken into account in equation (3). This equation is created for all pairs of transitions (tasks) T_j and T_k such that the output place P_i of the transition T_j is the input place of the transition T_k. Constraint for start times of tasks is considered only if both tasks are present in the final schedule, i.e. $v_j = 1$ and $v_k = 1$, constant *UB* ensures satisfaction of the equation otherwise. Equations (4) and (5) represent processors constraints for all pairs of tasks that are not joined by precedence constraints. These equations ensure that only one task is processed by the processor in a time. Constraint (6) represents limitation for number of input and output arcs that can be in the final schedule for each place. Number of output arcs present in the final solution is equal to number of input arcs present in the final schedule. Parameter *placeType* ensures that just one output arc of the source place and one input arc of the sink place will be present in the final solution and therefore at most one input/output arc can be present in the final solution for each place in a Petri net. Constraint (7) denotes that all input and output arcs of each transition have the same value of variable determining their presence in the final solution. Equation (8) determines that the transition T_i is present in the final solution if and only if all input and output arcs are also present. Equation (9) stands for evaluation of optimality criterion C_{max}; only completion times of tasks that are present in the final schedule are taken into account.

4.2.2. Infinite Amount of Identical Parallel Processors

ILP model, considering infinite amount of identical parallel processors, is identical to the ILP model with one processor. Only processors constraints are not taken into account, so the equations (4) and (5) are not considered.

4.2.3. Dedicated Processors

The problem with dedicated processors is close to the problem with one processor, mentioned in the previous text, except the processor constraints. Therefore, equation (4) and (5) are modified to equations (10) and (11). Equations themselves are the same as in the case with one processor, but they are created only for tasks assigned to the same processor R.

$$p_j \quad \leq \quad s_i - s_j + UB \cdot x_{i,j} + UB \cdot (2 - v_i - v_j)$$
$$i, j = 1, 2, \ldots, n; k = 1, 2, \ldots, m; i < j \text{ and } W[k,i] > 0 \text{ and } W[k,i] \neq \pm W[k,j] \text{and } R_i = R_j \qquad (10)$$

$$s_i - s_j + UB \cdot x_{i,j} \quad \leq \quad UB - p_i + UB \cdot (2 - v_i - v_j)$$
$$i, j = 1, 2, \ldots, n; k = 1, 2, \ldots, m; i < j \text{ and } W[k,i] > 0 \text{ and } W[k,i] \neq \pm W[k,j] \text{and } R_i = R_j \qquad (11)$$

Chapter 5

5. Utilization of the Simulation and Visualization

Areas of the simulation and visualization use in scheduling will be described in this section and some examples will be shown. Basic summary of this theme was shortly mentioned in Chapter 1. Both simulation and visualization have to be designed just for each case separately. There are some examples to illustrate utilization of the simulation and visualization in the following text. Some concrete implemented problems will be shown in Chapter 7.

5.1. Application of the Simulation

Scheduling of digital signal processing (DSP) algorithms is a practical example where the simulation can be used. Let us slightly describe the function of the Digital State Variable Filter (http://www.earlevel.com/Digital%20Audio/StateVar.html) [28], [29] that is used for example in processing of acoustic signals. This filter arises from the authentic analog version and its transcription results in the set of mathematical equations that represent filtering operations. Function of the filter is then realized by repeating of those operations in a never-ending loop. One of the reasons for the utilization of the digital version is easier design of the filter and easier modifiability as well. Role of the scheduling in the design of digital processing units, including filters, is to allocate given operations to one or more processors in time. From the scheduling process point of view, each mathematical operation corresponding to one task T_i and time needed for execution of this operation corresponds to processing time p_i. The goal of the scheduling algorithm is to create a time schedule containing all tasks while optimizing the given criterion. In the case of digital filter design, objective is to minimize the cycle time (period) of the schedule [27], [29]. Digital state variable filter is formed by the set of equations that are summarized below.

```
for $k = 2 : K$
        $FB(k) = F_1 \cdot B(k-1)$          // $T_1$
        $L(k) = L(k-1) + FB(k)$          // $T_2$
        $QB(k) = Q_1 \cdot B(k-1)$          // $T_3$
        $IL\{k\} = I(k) - L(k)$          // $T_4$
        $H(k) = IL(k) - L(k)$            // $T_5$
        $FH(k) = F_1 \cdot H(k)$          // $T_6$
        $BK = FH(k) + B(k-1)$            // $T_7$
        $N(k) = H(k) + L(k)$             // $T_8$
end
```

Applied attributes:	k	- current iteration of the algorithm.
	K	- number of iteration.
	I	- input value.
	L	- output value.
	F_1, Q_1	- constants.
	FB, QB, IL, H, FH, B, N	- discrete states; initially equal to zero.

Each mathematical operation is assigned to one task whereas addition takes one clock cycle of the processor unit and multiplication takes three clocks. Precedence constraints result from the relationship of operations used in the equations (see Figure 5.1).

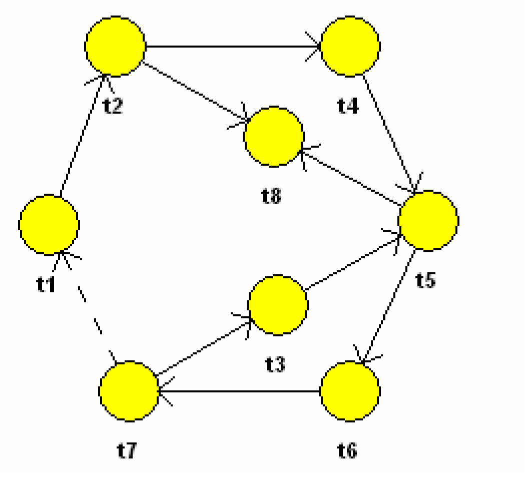


Figure 5.1 - Precedence constraints for DSVF filter

Two discrete time moments are important for the scheduling process for each task. First of them is the beginning of the task when all input data for an appropriate equation are fetched and the second one is the end of the task when all computed data are uploaded to the output (see Figure 5.2). Some computation is executed between these moments and data are being updated.

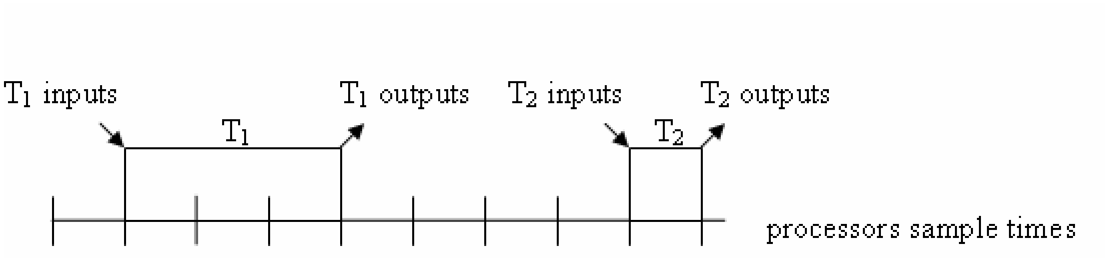


Figure 5.2 - Tasks execution

Each operation (task) must be executed just once during one iteration of the algorithm. Result of the simulation of DSVF filter with sample time equal to 220 kHz is displayed in Figure 5.3. The result was acquired using TrueTime [8] tool in Matlab.

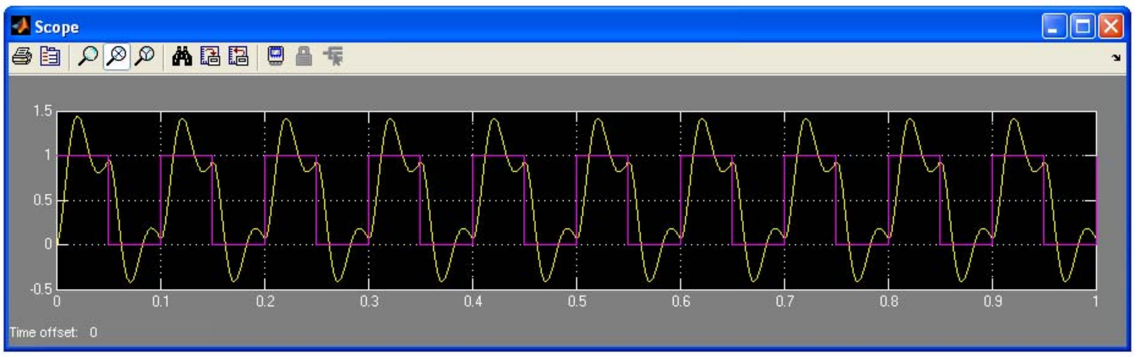


Figure 5.3 - Result of the simulation

Not only filtering of signals can be mentioned as utilization of the simulation in digital signal processing. *Model-based Predictive Control* (MPC) [30], [31] is another representative example of optimizations in DSP. MPC controllers are intended for control of linear (or linearized) systems subject to optimization of specified optimality criterion. An advantage of MPC control is an occasion to cover up wide spectrum of constraints and restrictions like limits for the value of control signals or for their rate of change in time. The most frequently used demand is to observe the reference signal. Design of MPC controllers arises from the state space equations of the system. The optimality criterion is formed by linear combination of energy used to control and squared difference between the required and real output. The goal of the MPC controller is to find a sequence of discrete values of control signal minimizing the criterion. Number of computed control signal values is called *prediction horizon*. There are more control strategies with MPC controller. One strategy is to compute the control sequence only on basis of state space model of the system and then to apply this control without regard to actual state of the system. Other strategy, closer to application of the simulation, is based on recalculation of the control sequence after each discrete step in time.

This strategy is called *receding horizon* because in each recalculation, new prediction horizon is computed, so the end of prediction is being shifted.

Solution of the MPC control problem can be found in numerical version, formed by the set of mathematical operations that represent matrices operations. These equations can be separated to elementary mathematical operations and then they can be assigned to tasks and scheduled with the given set of processors as in the case of DSVF filter. Basic idea of MPC control is displayed in Figure 5.4.

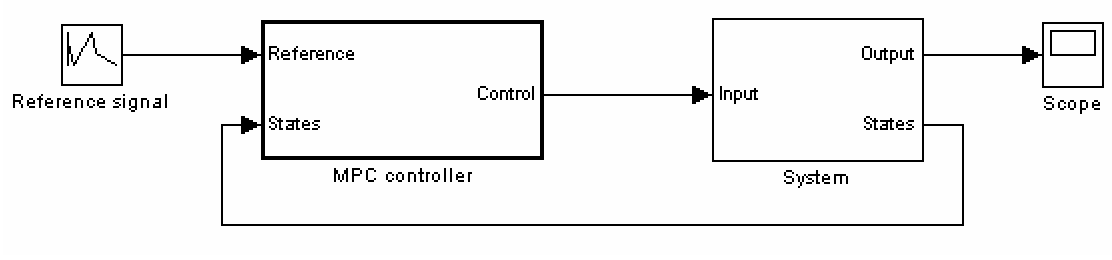


Figure 5.4 - MPC control

Reference signal and discrete states are brought to the block with model-based predictive controller. The actual computed value of control signal is at the output of the controller, which is connected with the controllable input (or more inputs) of the system. Function of the MPC controller can be tested and optimized via computer simulation. If the strategy with receding horizon is used, sequence of control signal values is computed in every sample time of the controlled system. The situation is the same as for DSVF filter simulation; to update one sample of the output signal, a set of operations has to be executed so the processor (or more processors) must have sufficient frequency to perform given operations in required time. In the real world cases, mathematical operations are assigned to processors in time via some scheduling strategy. Therefore, the simulation may take place in MPC control as well.

5.2. Application of the Visualization

The visualization serves as a tool to gain a better idea about realization of some problem in the first place. In the scheduling area, visualization can also serve as an optimization tool or as a verification for already scheduling problems. This verification is necessary e.g. for the production scheduling where not only time constraints has to be satisfied but also constraints caused by the adjustment in the space are important. These restrictions due to localization and movement of the material or machines are hard to cover up by the mathematical description and therefore hard to be taken into account for the scheduling process. A representative example is the hoist scheduling problem described in Section 2.2. The goal of the scheduling algorithm for the hoist scheduling problem is to schedule moves of one or more hoists with material, which is processed in several tanks with liquid. Figure 5.5 displays the basic idea of the hoist scheduling problem. In some cases, the load and the unload station can be merged into one load/unload station.

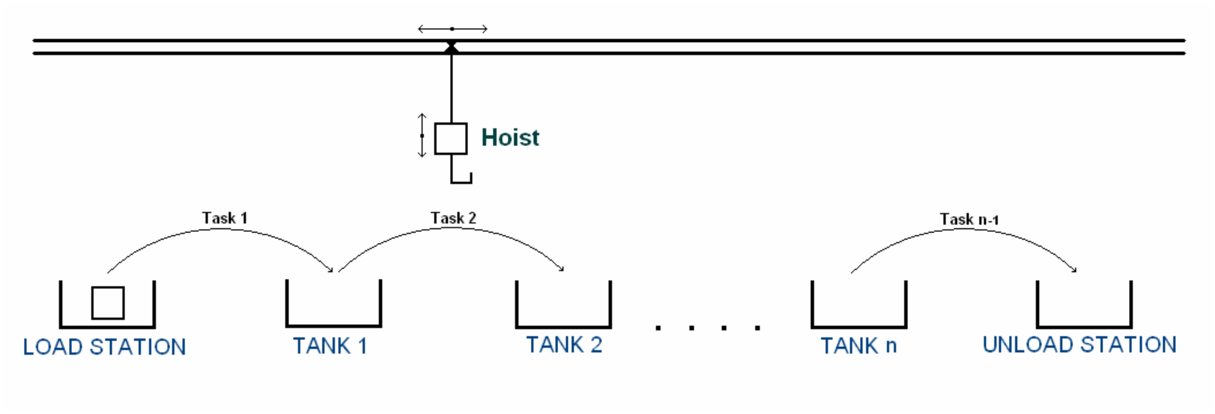


Figure 5.5 - Hoist scheduling problem

Visualization of the hoist scheduling problem can prove feasibility of the given time schedule or detect new restrictions that must be involved in the scheduling algorithm. Space restrictions are important especially in situation with more hoists when their trajectories may intersect.

Another area for visualization of production activities are *process flows*. For the purpose of production cost reduction and productivity of work growth, amount of activities that do not yield economic profit has to be minimized. As a consequence, production operations that do not add a value to the product have to be eliminated or suppressed at least. Material transfers, machine setup times, tool handovers and cleaning are examples of activities to be avoided. Process flow is two or three-dimensional transcript of the production plan. It is formed by lines and curves that connect machines, buffers and other stations in the graphical approach of the production plan. Diagram of the movement in the space is very important for projecting the allocation of machines, buffers, tool repositories and other stations. This type of the visualization plays the role during the planning of production plans and it can be also used for representation of the already scheduled solution. An example of the process flow is depicted in Figure 5.6. Each object with label *M* represents one machine for processing of the material and each object with label *B* represents one buffer for temporary storage of the material. Solid curves stand for material transfers, dashed curves represent moves of required instruments and dotted curves stand for garbage collection.

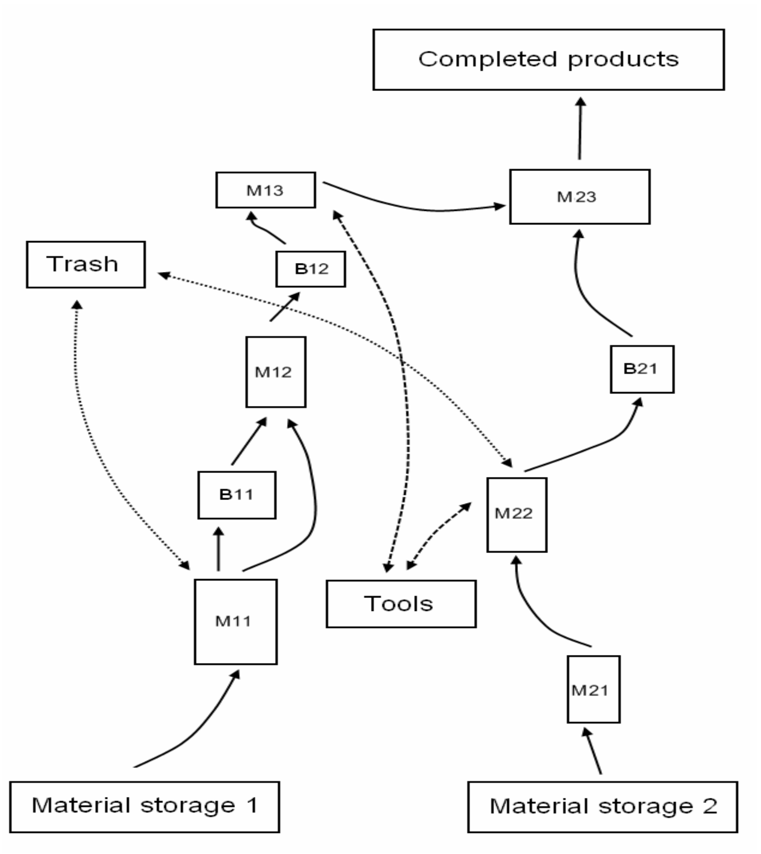


Figure 5.6 - Example of process flows

Chapter 6

6. Implementation of VISIS

As mentioned in Chapter 1, VISIS is intended to be a tool for the simulation and visualization of the scheduling results. VISIS is implemented in the Matlab environment using data structures and algorithms from TORSCHE Scheduling Toolbox for Matlab [7] and it will be a part of this toolbox in next release planned on October 2008. Users of VISIS can define their own project in the Virtual Reality toolbox, standard part of Matlab, and bind this definition with Matlab commands. Both simulation and visualization are then realized in Simulink, which is also a default part of the Matlab environment. The VISIS implementation provides several functions available for users and there are also several supplemental functions. In order to maximum simplicity of usage, resulting Simulink model is generated automatically. This output model contains one masked subsystem representing the control system. In case of the visualization, there is another block referencing to the predefined virtual reality world. The mask of the control subsystem has inputs and outputs with user-defined names and sizes. The core of the control subsystem is the S-Function block, which contains main control function. This function realizes updating of outputs according to the given schedule and actual values of inputs. This control function is also generated automatically and all needed external data are created in Matlab workspace before the start of the simulation. The S-Function block has only one input and output port as default so the in/out signals are integrated/divided to reach user-defined number of inputs and outputs. This subsystem is then masked as one block with appropriate ports. The Simulink model and code for the S-Function block are both generated as text files from the prepared templates. The control function is called for each sample of Simulink and the outputs are updated according to the schedule and actual Simulink time.

6.1. Implemented Functions

All functions implemented during realization of the VISIS application are standard Matlab m-files. In this subsection, we present all of them with description of input and output variables and command syntax.

adduserparam - function that loads commands form the text file and assign them to tasks due to the format of the given text file. Commands from the text file are stored in attribute *UserParam* for each task. Some parameters can be also stored in attribute *TSUserParam* of the taskset.

Syntax:

$TSout = addcode(TS, file)$

Where *TS* is a taskset object, *file* is a string with the name of the text file and *TSout* is output taskset with assigned commands. In the text file, each task has to be introduced by its identifier which can be its name (property *Name* of the task) or by special character # and then ordinal number of a task in the taskset. Following set of commands is assigned to an appropriate task up to the keyword *endparam*. After this keyword, new set of commands introduced by new identifier of a task can be written. Commands to be stored to attribute *TSUserParam* of the taskset have to be introduced by the keyword *UserParam.begin* and terminated by the keyword *endparam*. Detailed utilization of this function is described in Appendix A. All commentaries (introduced by the character *%*) inside the task definition are copied to the S-Function and all other are ignored.

setports - function that serves for setting the name and size of ports of the control block in Simulink.

Syntax:

$ports = setports(varargin)$

Where *ports* is the output structure containing information about user-defined names and sizes. Input of this function is alternative and its length depends on the amount of inputs and outputs that are needed for the concrete application. Setting of inputs starts with keyword *Input* and arbitrary amount of inputs can be listed in the form (*input_name*, *size_of_input*). Size of input represents length of the vector corresponding to an appropriate port. The setting of outputs is the same, only with the keyword *Output* now. An example of the function call is in Appendix A again.

VRcontrol - function for setting of objects of virtual reality and their properties that will be controlled from Simulink. Inputs of the virtual reality block are automatically created concerning this definition.

Syntax:

$VRin = VRcontrol(varargin)$

Where *VRin* is an output structure containing information about user-defined names and properties of the virtual reality (VR) objects. As an input for this function, one or more pairs of the exact VR object name and its property can be defined in the form (*object_name, object_property*). Only numerical parameters of VR objects and strings of text boxes can be controlled from the Simulink model.

taskset2simulink - main function of VISIS. This function generates the Simulink model and all other files and data structures needed for the simulation and visualization.

Syntax:

taskset2simulink(file, TS, ports, VRin, stopTime, varargin)

Where *file* is a string with name of the virtual reality file that must be terminated by the suffix *.wrl*. If the VR is not used, arbitrary name of project can be set or empty parameter *[]* can be passed to the function and all created data will have default prefix *project1*. *TS* is the taskset object with included schedule and Matlab commands assigned to tasks. Structure *ports* is the parameter that can be obtained by the *setports* function and *VRin* is the structure resulting from the *VRcontrol* function. Parameter *stopTime* serves to set duration of the simulation or visualization in Simulink. Some additional information can be passed to the function via arbitrary input. User of VISIS can set the sample time of the simulation (default value is equal to one), period of schedule repeating and it is also possible to generate only the control function instead of the whole Simulink model. Detailed description of input parameters is in the Appendix A.

The main function *taskset2simulink* handles arbitrary inputs, if any exists, in the first place. To be available for the resulting Simulink model, four variables are assigned with their names in the workspace of Matlab - *file*, *period*, *TS* and *sampleTime*. The code for the control S-Function is then generated and if there is any structural error in the created code, a warning is displayed. The last step of the *taskset2simulink* is to generate the output Simulink model and if there is no error, the model is opened.

sfunctioncode - supplemental function, which generates code for the S-Function regarding to the text assigned to tasks. The function is called from the main function *taskset2simulink* and a new m-file is saved to the current directory of Matlab.

Syntax:

sfunctioncode(file, TS, ports, VRin, dispVR)

Where *file* contains the name of the project, *TS* is the taskset object with assigned code, *ports* is the structure resulting from the *setports* function and *VRin* is the structure resulting from the *VRcontrol* function. Parameter *dispVR* is a binary variable that determines if the virtual reality is used or not. Function generates code for the S-Function as a string that is inserted to a new text file afterwards. This file is saved as *S_projetName.m* where *projetName* is the name defined by user or *project1* as default. For the generation of the S-Function code, predefined template file is used, one for the case with virtual reality (file *SFunctionBase.m*) and one for the case without VR (file *SFunctionBase2.m*). Some text from the template is copied and the rest is added according to size of taskset and commands assigned to tasks.

parsetask - supplemental function that loads data from the task attribute *UserParam* and transforms them to the form for the S-Function code.

Syntax:

parsetask(T, index1, index2)

Where *T* is a task object, which contains code to be transformed, *index1* is the index of start time of task *T* in the vector of discrete states of the S-Function (will be described later) and *index2* is the index of completion time of task *T* in the vector of states. This function is called from the *sfunctioncode* for each task in the taskset.

simulinkmodel - supplemental function serving to generate the output Simulink model. The model is generated as a text file from the predefined template files. The function is called from the main function *taskset2simulink* and a new Simulink model file is saved to the current directory of Matlab.

Syntax:

simulinkmodel(file, *stopTime*, *sampleTime*, *ports*, *VRin*, *dispVR)*

All input parameters have the same meaning as for the previous functions. The function generates system with one S-Function block, one *mux* (multiplexer) block and one *demux* (demultiplexer) block. Amount of ports for both the mux and demux blocks are adjusted with regard to count of user-defined inputs and outputs. The S-Function block in Simulink has just one input and one output port, so the signals have to be integrated/divided to satisfy given demands for count and names of the inputs/outputs. The whole system is then masked as one subsystem with appropriate ports. If the virtual reality is also defined, another block (*VRsink*), referencing to the given VR file, is also added to the model. Simulink model is generated as a string and it is inserted to a new text file afterwards. This file is saved as *projetName.mdl* where *projetName* is the name defined by the user or *project1* as default.

getVRpar - function that allows to obtain any numerical or string value of the objects from the virtual reality represented by the VRsink block in Simulink.

Syntax:

value = getVRpar(file, *object*, *var)*

Where *file* contains the name of the project, *object* is a string with the exact name of the object in the VR file, *var* is the name of the parameter to acquire the value from and *value* is the value of required parameter. The *getVRpar* function serves mainly to get values of parameters that are not directly controlled from the Simulink model. Parameters, which are controlled via inputs of VR block in Simulink are available in the S-Function under the same name.

isin - supplemental function that is used in the S-Function code to recognize if the following set of commands is to be executed in the current time.

Syntax:

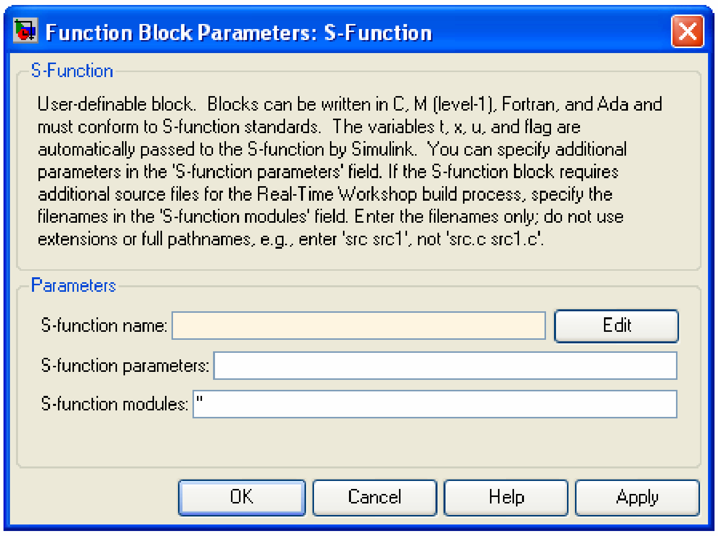
status = isin(start, stop, period, t)

Where *start* and *stop* are the numerical values representing start time and completion time of a task in the schedule, *period* is the period of the schedule and *t* is the current time in Simulink. Output binary variable *status* is equal to one if and only if a task determined by its start time and completion time is to be executed in current time respecting the period of the schedule.

Functions *adduserparam*, *setports*, *VRcontrol*, *taskset2simulink* and *getVRpar* are intended for direct work with VISIS whereas *sfunctioncode*, *parsetask*, *simulinkmodel* and *isin* are supplemental functions not mentioned to be used by users of VISIS.

6.2. Simulink Model Description

Both the simulation and visualization of scheduling results are realized in Simulink, which is a part of Matlab intended for wide spectrum of simulations. The Simulink model is generated automatically and it contains one control block with included *S-Function* block and in the case of visualization, there is also one *VR sink* block referring to the given file with the virtual world definition. The S-Function block is intended for executing of Matlab commands, written in standard Matlab m-file, during the simulation in Simulink. Inputs for the S-Function block are the name of the assigned m-file, names of variables in the workspace to be passed to the S-Function and the last input is definition of additional modules. Dialog box for the S-Function block is depicted in Figure 6.1. The S-Function block itself has one input port and one output port. Length of the vector that will be accepted in the input of the block and the vector length of the output are defined in the assigned m-file. Input vector is available under the name *u* in the S-Function and more, actual Simulink time is stored in variable *t*, vector of internal states is in the variable *x* and *flag* is automatically updated variable, which determines actual step in the S-Function call.


Function Block Parameters: S-Function

S-Function

User-definable block. Blocks can be written in C, M (level-1), Fortran, and Ada and must conform to S-function standards. The variables t, x, u, and flag are automatically passed to the S-function by Simulink. You can specify additional parameters in the 'S-function parameters' field. If the S-function block requires additional source files for the Real-Time Workshop build process, specify the filenames in the 'S-function modules' field. Enter the filenames only; do not use extensions or full pathnames, e.g., enter 'src src1', not 'src.c src1.c'.

Parameters

S-function name:

S-function parameters:

S-function modules: ''

Figure 6.1 - S-Function dialog box

Execution of the S-Function is separated into several steps for each function call in dependence on the type of internal states and required sample time of operations. Decision about next routine to be executed is made on basis of the actual value of the *flag* variable. If the S-Function block is called for the first time, initialization routine is executed. During this routine, number of continuous and discrete states, number of inputs and outputs, sample time of the function and initial values of the internal states are defined. After the initialization, subfunction for updating output values is called. If there are any continuous states (not in case of VISIS implementation), next step of the function is to compute their derivates. Discrete states are updated after the continuous derivates computation and if the variable sample time of the S-function is defined, next time moment for the S-Function block calling is computed. After the last step of the simulation, termination routine is executed. All these subfunctions except the initialization and termination are executed during each call of the S-Function.

VISIS implementation involves only discrete states and the sample time is fixed, so the steps for derivates and next calling time computation are skipped. In the initialization, number of discrete states, inputs and outputs are automatically computed in regard to given taskset and user-defined input and output ports. Start times, processing times and processors specification are read from the taskset and saved to the vector of internal states. Values of the controlled inputs of virtual reality are assigned to appropriate output variables and saved to internal states too.

User-defined code assigned to tasks is included in the subfunction for updating discrete states. Current discrete step is computed from the actual time of the Simulink simulation and current values of controlled inputs of the virtual reality block are assigned to appropriate variables via function *getVRpar*. Function *isin* is used to determine which task is executed in current step of the simulation in regard to the given time schedule. Tasks can be divided into more parts (see Appendix A), so the decision is made for each part of each task separately. At the end of discrete states updating, all computed values are stored to predefined positions in the vector of internal states, so all these values will be available in the next step of the simulation. The last subfunction, called in one sample of the S-Function block, updates outputs. An appropriate part of the vector of internal states is copied to the output vector in this subfunction.

The S-Function block is connected with requested number of inputs and outputs and the whole system is then masked as one subsystem in Simulink. This mask is called *projectName_Subsystem* where *projectName* is a name specified by user of VISIS and the inputs and outputs of this masked subsystem correspond with user definition. Example of the generated subsystem is displayed in Figure 6.2 and its mask is displayed in Figure 6.3.

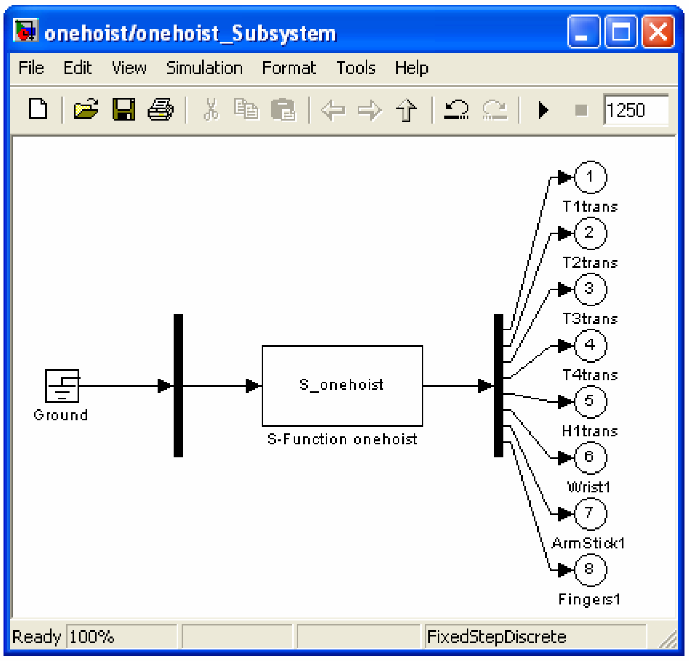


Figure 6.2 - Subsystem with S-Function block

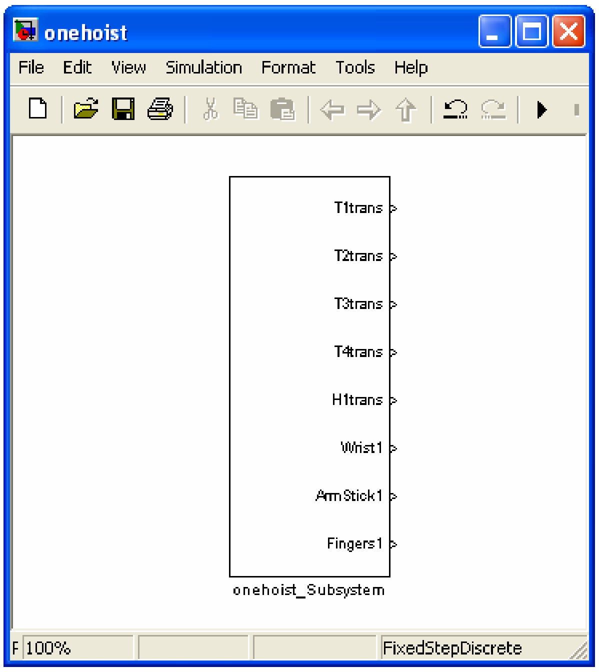


Figure 6.3 - Mask of the subsystem

As we can see in Figure 6.2, any unused port is terminated by the ground to avoid warnings and errors caused by not connected ports. All Simulink objects are generated automatically and their sizes, positions and number of ports are also set automatically. If the virtual reality is used, block referring to specified VR file is also included. Figure 6.4 displays the whole Simulink model, which is a result of *onehoist_demo* example.

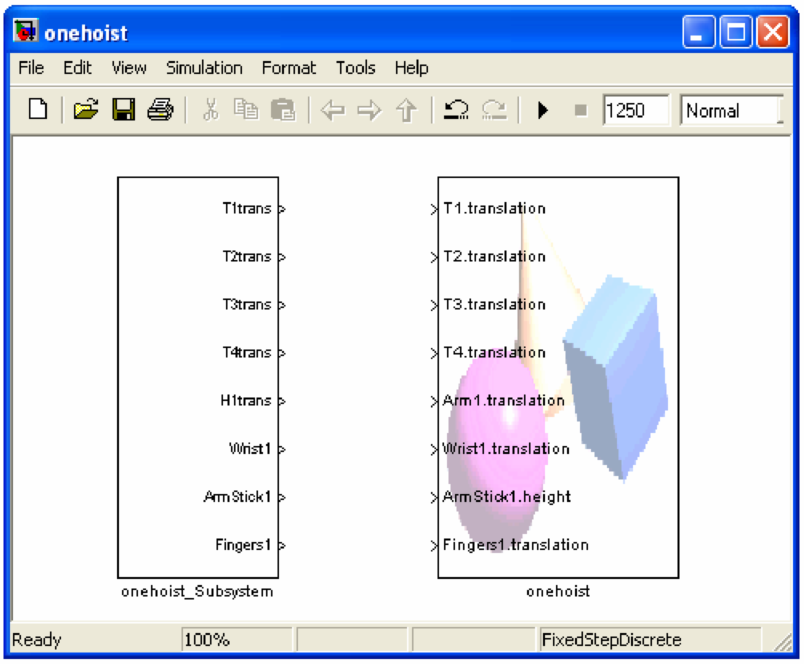


Figure 6.4 - Simulink model with VR block

It is possible to add any other object to the generated Simulink model and then start the simulation. Virtual reality world is automatically opened in the case of visualization.

6.3. Task Representation

Each task is determined by its processing time and start time in the schedule. Completion time of a task can be easily computed from these values. All operations defined for a task are executed in regard to these numerical values. To allow as much precise problem representation as possible, each task can be separated to more parts with different commands and different duration. Each part of a task is relative to its start time in the schedule. There are three ways how to define operations for one task in time (see Appendix A). Commands assigned to a task can be repeated for a defined number of samples or executed just once. In regard to Figure 5.2 we can define operation of loading data executed in the time moment corresponding with the start time of a task, operation of uploading data in the moment corresponding to completion time and any repeating operation executed for every sample of the Simulink simulation between these time moments.

6.4. Virtual Reality Toolbox

Graphical objects for the visualization are created in *VRedit* (part of Virtual Reality toolbox for Matlab). VRedit allows to define basic geometrical objects, text, background, textures and complex objects. VR toolbox links MATLAB and Simulink with virtual reality graphics, enabling MATLAB or Simulink to control the position, rotation, dimensions, etc. of the 3-D images defined in the virtual reality environment. To be controllable, the object in virtual reality must have unique name. This identifier has to be chosen as an input of the VR block in Simulink together with the property that will be controlled. In each sample of the Simulink simulation, properties of the virtual reality world are updated according to the vector values at the input of VR block. VISIS generates the whole Simulink model so the input ports for VR block are defined automatically.

To change string value in the virtual reality world, function *setfield* can be used. Parameters of this function are the virtual reality node, its property and new requested value. Example of setting the string value of the node *Tank1* in the VR file *onehoist.wrl*:

```
w=vrworld('onehoist.wrl');
open(w)
node = vrnode(w,'Tank1');
setfield(node,'string','Busy')
close(w)
```

Chapter 7

7. Case Studies

In this section, performance measures for the new algorithm for the optional scheduling problem are presented. Random generator of instances for the problem with alternative process plans was realized created and the algorithm was tested using these data. Furthermore, four examples for the visualization and one for the simulation with VISIS were created to show capabilities of VISIS and three of them are described in this section.

7.1. Performance Measures for the Optional Scheduling Problem

There are three algorithms proposed for the optional scheduling problem; one for the problem with one processor, one for infinity amount of identical processors and one for the case with dedicated processors. For performance measures, algorithm for the problem with one processor is used. Figure 7.1 displays mean CPU time used to solve the problem in dependence on number of transitions in a Petri net. All displayed results are average from 100 measurements with randomly-generated instances. These instances consist of Petri net incidence matrix *W* and vector of processing time *p*. The incidence matrix *W* results from a Petri net that represents instance of the problem with alternative process plans. The implemented generator of instances creates one source place with number of output transitions uniformly generated from the vector $v_1 = [2,2,2,2,2,3,3,3,4,4]$. Number of places is then uniformly generated from the vector $v_2 = [1,1,1,1,1,2,2,2,2,3]$ and each transition is randomly connected with one output place. For each place, number of output transitions is uniformly generated from vector v_1 and these steps are repeated until the specified amount of transitions (tasks) is acquired. At the end, all transitions without output place are connected to the sink place. Each processing time is generated uniformly from the interval <1;10>. The measurement was performed for the amount of tasks from 5 to 28. Time complexity of the presented ILP model is exponential.

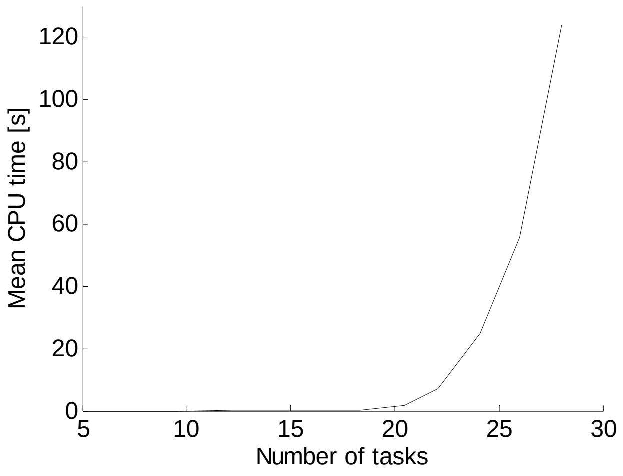


Figure 7.1 - Mean solving time

Figure 7.2 displays dependence of memory size used for solving on the amount of tasks. Solving demands for memory are very low in comparison with the time complexity.

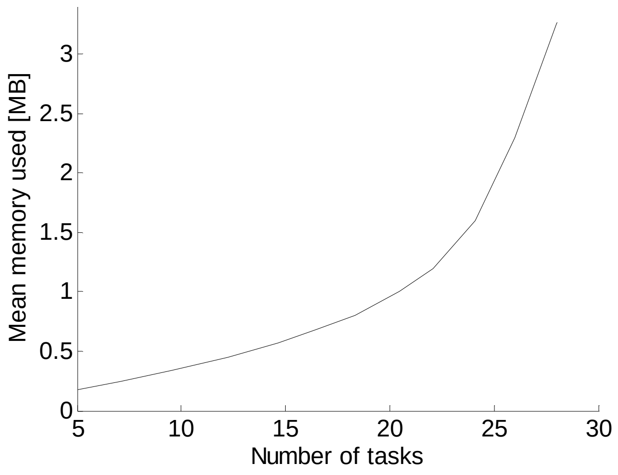


Figure 7.2 - Mean memory used

Number of variables and constraints in dependence on size of input problem is displayed in Table 7.1.

Variables	binary	$n + n \cdot m + n^2$
	integer	n
Constraints		$m \cdot n \cdot (m + n)$

Table 7.1 - Number of variables and constraints

Where *n* is amount of PN transitions and *m* is amount of PN places.

Figure 7.3 shows for a given amount of time, how many instances have been solved. Measurement was realized for three quantities of tasks with one hundred instances for each quantity.

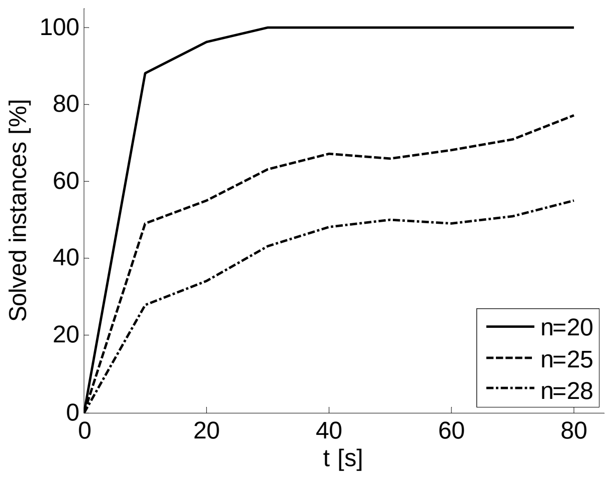


Figure 7.3 - Ratio of the solved instances in time

All measurements was realized on computer with 1 GB operational memory and processor dual-core 1.6 GHz.

7.2. Simulation with VISIS

Simulation of Digital State Variable Filter (DSVF), described in Chapter 5, was implemented in Simulink using VISIS application. This filter is formed by the set of equations with elementary arithmetic operations. Each operation is assigned to one task and in the text file, it has the following form:

```
#1
in 0
x(8) = 0.0079 * x(7);
in 3
x(1) = x(8);
endparam
```

This part of script denotes that data for the operation are loaded in the beginning of first task of the taskset in the schedule and the resulting output is available after three samples of task executing. All other operations are assigned to tasks similarly. Input for the filtering is stored in variable *I* and the output variable is named *L* in the commands for tasks. All other variables are stored in the vector of internal states. Matlab commands for the definition of simulation with VISIS are written below.

```
%Define taskset and add code for tasks
TS = taskset([3 1 3 1 1 3 1 1]);
TS = adduserparam(TS,'dsvf.txt');

%Define period of tasks
period = 11;

%Set the schedule
starts = [0 4 3 5 6 7 10 7];
add_schedule(TS,'dsvf',starts,TS.ProcTime)

%Define parameters for the simulation in Simulink
stop = 1;
sample = 1/220000;

%Define inputs and outputs for the S-Function block
ports = setports('Input','I',1,'Output','L',1);

%Call main function
taskset2simulink('dsvf',TS,ports,[],stop,'Period',period,'Sample',sample);
```

The Simulink model resulting from this simple definition contains one masked subsystem with input *I* and output *L*. Pulse generator with unit amplitude and period 0.1s and the Scope unit are added to the model and the simulation is ready to start, see Figure 7.4.

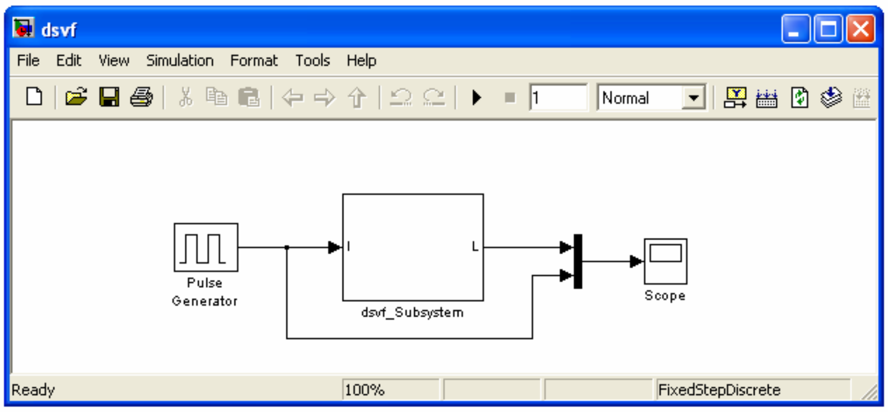


Figure 7.4 - Simulink model of DSVF

The result of the simulation is the same as in case with TrueTime tool, described in Chapter 5. The input and output signal of the simulated filter is displayed in Figure 7.5; purple line is the signal from the pulse generator and the yellow one is the filtered signal.

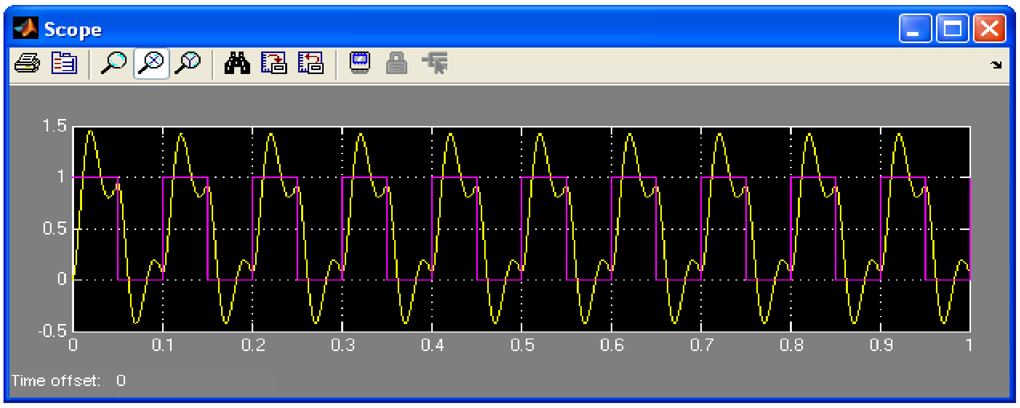


Figure 7.5 - DSVF simulation signals

7.3. Visualization with VISIS

Technique for the visualization is very similar to the case of the simulation with only one difference during definition. Ports for the VR block in the Simulink model have to be defined and the virtual reality world has to be created indeed.

The first realized example is the visualization of the hoist scheduling problem, described in Chapter 5. More precisely, two visualizations for the hoist scheduling problem were created, first one with one hoist carrying material and the second one with two hoists. The material has to be processed in three tanks with liquid in both cases. For the situation with only one hoist, load and unload stations are merged into one place. Graphical appearance was defined in the VRedit environment and the project is designed as a 2D visualization. Controllable properties are positions of the material (represented by square objects with different textures), horizontal positions of the hoists, lengths of hoist arms and vertical positions of hoist wrists. Moreover, string values representing the amount of waiting and finalized material are being changed using function *setfield*. The Simulink model for the case with one hoist is displayed in Figure 6.4 and the initial state of virtual reality is in Figure 7.6.

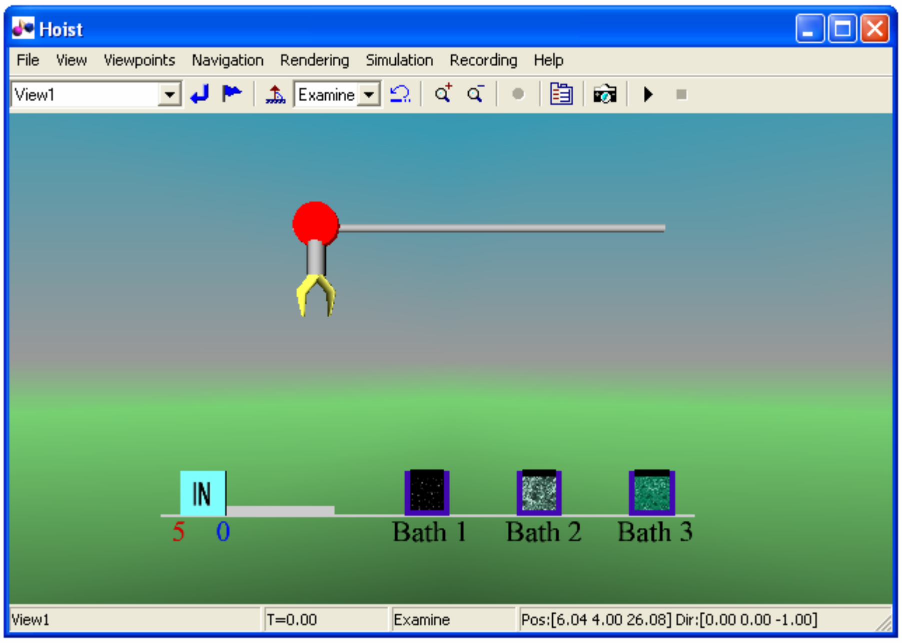

Hoist
File View Viewpoints Navigation Rendering Simulation Recording Help
View1
Examine
IN
5 0
Bath 1
Bath 2
Bath 3
View1
T=0.00
Examine
Pos:[6.04 4.00 26.08] Dir:[0.00 0.00 -1.00]

Figure 7.6 - Virtual reality for the Hoist scheduling

Four tasks are needed to represent moves of the hoist with the material. The schedule with these tasks is repeated periodically and the commands to perform the moves are executed in regard to start time of an appropriate task in the schedule. Progress of the visualization is slightly demonstrated in Figure 7.7.

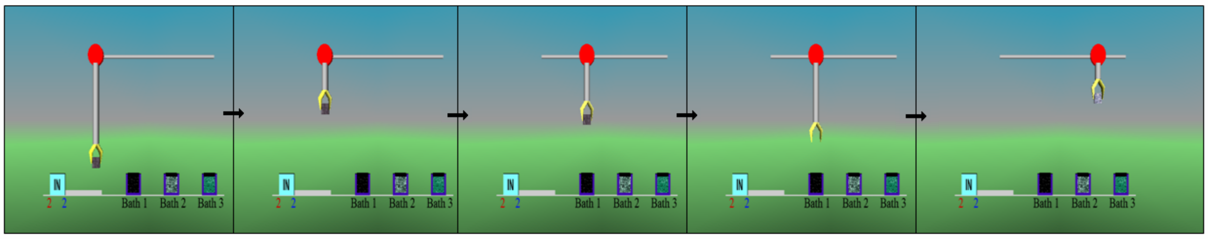


Figure 7.7 - Progress of the visualization

Second example of the visualization with VISIS is the workshop for production of small lamps. How to modify position, size, rotation and also color of objects is shown in this example. One frame of the visualization is displayed in Figure 7.8.

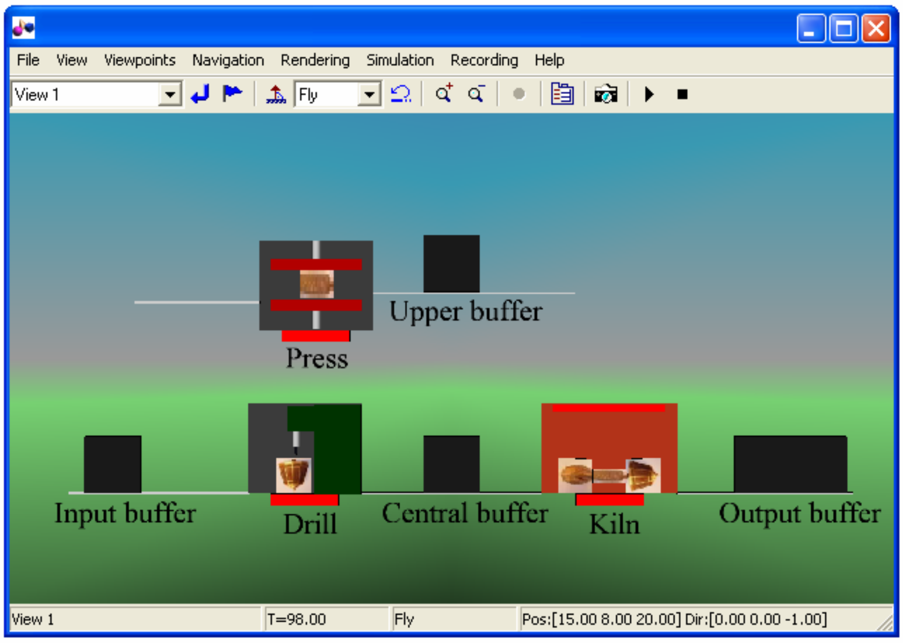


Figure 7.8 - Visualization of the workshop

The last realized visualization with VISIS is the motivation example to show importance of the scheduling. Let us imagine a river and four soldiers standing on the shore in the night. They have only one flash-lamp and to get on the other side of the river, at most two soldiers can walk together and they must have the flash-light. Each soldier can walk through the river with different speed and movement of two soldiers is made with the speed of the slower one. The goal is to transport all soldiers on the adverse side of the river in the shortest possible time. Movements of soldiers are represented by tasks with processing times equal to time needed to cross the river and two time schedules are created, one with the correct progress of soldiers transfers and one with the incorrect progress. Difference in acquired times can show the purpose of the time scheduling. One frame of the realized visualization for this problem is shown in Figure 7.9. Times needed for transports of soldiers are displayed on the left side and actual time value is displayed in the upper left corner.

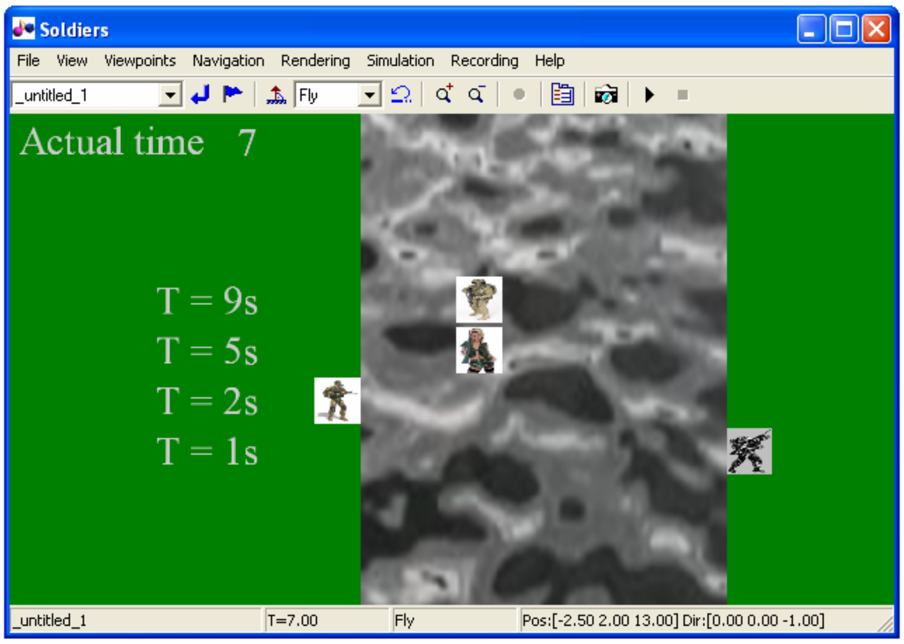

Soldiers
File View Viewpoints Navigation Rendering Simulation Recording Help
_untitled_1
Fly
Actual time 7
T = 9s
T = 5s
T = 2s
T = 1s
_untitled_1
T=7.00
Fly
Pos:[-2.50 2.00 13.00] Dir:[0.00 0.00 -1.00]

Figure 7.9 - Motivation example

7.4. Profiler Results

In this subsection, we will show some results acquired from the Matlab profiler, which supports measurement of execution time of functions. Profiler returns information about time spent by each function that is called during the selected command execution. Number of function calls and detailed list of called subfunctions are also available. For each created example, presented in the previous text, execution times of VISIS functions are measured and the results are stated in Table 7.2.

Function / Example	Filter	Hoist	Workshop	Soldiers
adduserparam	0,29 s	0,34 s	0,49 s	0,34 s
taskset2simulink	1,72 s	2,46 s	2,78 s	4,22 s
sftunctioncode	0,48 s	0,41 s	0,71 s	0,36 s
simulinkmodel	0,53 s	0,55 s	0,62 s	0,52 s
number of tasks	8	4	14	5

Table 7.2 - Execution times

To generate the Simulink model and the included S-Function, it takes only a few seconds and it does not increase rapidly in dependence on the amount of tasks or the length of the given text file. Most of the execution time of the main function *taskset2simulink* is spent by opening of the generated Simulink model and critical is the graphical complexity of the virtual reality file. The Simulink model file and the S-Function code are generated as strings and then included into the new file, so the time complexity of generation in dependence on amount of tasks and text file length is linear.

Distribution of time between functions during the simulation of the Digital State Variable Filter was probed. The S-Function is divided into several steps (see Chapter 6) and most of time (17 from total 26 seconds) of its execution is spent by the subfunction for updating discrete states. This is the expected fact, because all commands specified by user of VISIS are executed during this part of S-Function. Time needed to update discrete states is distributed uniformly through the function so there is no bottle-neck, which would halt the simulation. Simulation by VISIS needs approximately 80% of time in comparison with the same example realized using TrueTime library. In addition, time needed for one second of simulation with 220000 samples per second is approximately 32 seconds in TrueTime and 26 seconds in VISIS.

Chapter 8

8. Conclusions

This work presents results achieved in two parts of scheduling area. New algorithm for the optional scheduling problem has been proposed and VISIS, an application for the visualization and simulation in scheduling, has been realized in the Matlab environment. Both parts offers solution for problems that are usable in the area of production scheduling. This branch of the scheduling theory plays an important role nowadays and time savings during the production planning and scheduling lead to less demanding manufacturing of products.

For the implementation purpose of the new scheduling algorithm for the optional scheduling problem based on Integer Linear Programming (ILP), terminology arising from the related works has been established in the first place. Original representation by the Petri nets formalism is then proposed. This concept of input data assignment allows to define the problem structure naturally and also later modification of the structure is simple. The ILP model is designed from the state-transitions matrices of given Petri net. Proposed solution based on ILP solution was tested on random-generated data and the results show that the algorithm is suitable to solve optional scheduling problems with up to 30 tasks in within a few minutes with low memory requirements (only about 5 MB for the biggest tested instances). The algorithm is capable to solve the problems defined as a Petri net with specified properties. Processing time, release time and deadline can be assigned to each task. Proposed integer linear programming model can be easily extended, new task parameters can be added and the optimality criterion modified.

Established terminology will be a base for the following research in the area of the optional scheduling. New optimal algorithms can be proposed and some polynomial heuristic can arise from the properties of Petri nets. Moreover, utilization of Petri nets offers a possibility to use some of many existing methods for their analysis and simplifications.

VISIS has two areas of use: in discrete simulation (e.g. in digital signal processing) and in the visualization of scheduled problems. It is planed as an extension of future version of TORSCHE Scheduling Toolbox for Matlab. The application can be used for presentations, educational purposes or as an optimization tool and whenever clear presentation of results is needed. During the implementation of VISIS, several functions for work with this tool and several supplemental functions were realized. Some demonstrative examples was created to show the capabilities of VISIS. Visualization with VISIS supports user-defined appearance of the virtual reality and it allows to bind this definition with arbitrary Matlab commands. Up to our knowledge, there is no such a tool providing visualization of scheduled problems in that range. The simulation in scheduling can serve as a fast feedback for results of scheduling process. Simulation of digital state variable filter is faster than in TrueTime library since VISIS is optimized for simulations of time schedules. The main advantage of VISIS is easier problem definition and simple usage. The implemented application is designed to maximize the comfort and simplicity of utilization.

References

[1] R.BARTAK: Unary Resource Constraint with Optional Activities. In *Lecture Notes in Computer Science*, Vol. 3258/2004, p. 62-76.

[2] J.CH.BECK and M.S.FOX: Constraint-directed Techniques for Scheduling Alternative Activities. In *Artificial Inteligence*, 2000, Vol. 121, p. 211-250.

[3] A.WEINTRAUB, D.CORMIER, T.HODGSON, R.KING, J.WILSON and A.ZOZOM: Scheduling with Alternatives: a Link Between Process Planning and Scheduling. In *IIE Transactions*, 1999, Vol. 31, p. 1093-1102.

[4] C.SAYGIN, F.F.CHEN, J.SINGH: Real-Time Manipulation of Alternative Routeings in Flexible Manufacturing Systems: A Simulation Study. In *The International Journal of Advanced Manufacturing Technology*, 2001, Vol. 18, p.755-763.

[5] E.VIN, P.LIT, A.DELCHAMBRE: A Multiple-objective Grouping Genetic Algorithm for the Cell Formation Problem with Alternative Routings. In *Journal of Intelligent Manufacturing*, 2005, Vol. 16, p. 189-205.

[6] J.BLAZEWICZ et al: Scheduling in Computer and Manufacturing Systems. *Springer*, 1993.

[7] P.ŠŮCHA, M.KUTIL, M.SOJKA and Z.HANZÁLEK: TORSCHE Scheduling Toolbox for Matlab. In *IEEE International Symposium on Computer-Aided Control Systems Design*, 2006, p. 50-52.

[8] M.OHLIN, D.HENRIKSSON and A.CERVIN: TRUETIME 1.4—Reference Manual. Department of Automatic Control, Lund University, 2006.

[9] G.S.FISHMAN: Discrete-Event Simulation: Modeling, Programming, and Analysis. *Springer*, 2001.

[10] F.MANLIG and M.ŠRÁMEK: Řízení výrobních zakázek s podporou počítačové simulace. In journal *Průmyslovés inženýrství*, 2003, p. 119-123.

[11] J.MISRA: Distributed Discrete-Event Simulation. In *ACM Computing Surveys (CSUR)*, 1986, Vol. 18, p. 39-65.

[12] D.TOAL, T.COFFEY and P.SMITH : Expert Systems and Simulation in Scheduling. *CiteSeer*, 2000.

[13] H.WWTTSTEIN, H.ZOLLER and G.LIEFLANDER: Visualization of Process Scheduling. Universität Karlsruhe, Department of Computer Science. http://i30www.ira.uka.de/teaching/coursedocuments/processscheduling/

[14] A.SCHRIJVER: Theory of Linear and Integer Programming. *Paperback*, 1998.

[15] T.MURATA: Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, 1989, Vol. 77, p. 541-580.

[16] W.E.WILHELM: A Column-Generation Approach for the Assembly System Design Problem with Tool Changes. In *International Journal of Flexible Manufacturing Systems*, 1999, Vol. 11, p. 177-205.

[17] R.HELIMANN: A Branch-and-bound Procedure for the Multi-mode Resource-Constrained Project Scheduling Problem with Minimum and Maximum Time Lags. In *European Journal of Operational Research*, 2003, Vol. 144.

[18] G.TUNCEL, G.M.BAYHAN: Applications of Petri Nets in Production Scheduling: a Review. In *The International Journal of Advanced Manufacturing Technology*, 2007, Vol. 34, p.762-773.

[19] G.MEJIA, C.MONTOYA: A Petri Net Based Algorithm for Minimizing Total Tardiness in Flexible Manufacturing Systems. In *Annals of Operations Research*, 2007.

[20] R.L.GRAHAM, E.L.LAWLER, J.K.LENSTRA, A.H.G.RINNOOY KAN: Optimization and Approximation in Deterministic Sequencing and Scheduling Theory: a Survey. In Annals *of Discrete Mathematics*, 1979, Vol. 5, p. 287-326.

[21] P.BAPTISTE, C.LE PAPE, W.NUIJTEN: Constraint-Based Scheduling - Applying Constraint Programming to Scheduling Problems. *Springer*, 2001.

[22] J. W. HERRMANN: Handbook of production scheduling. *Springer*, 2006.

[23] H.KELLERER and V.A.STRUSEVICH: Scheduling Problems for Parallel Dedicated Machines under Multiple Resource Constraints. In *Discrete Applied Mathematics*, 2003, Vol.133, p.45-68.

[24] M.A.MANIER and CH.BLOCH A Clasification for Hoist Scheduling Problems. In *International Journal of Flexible Manufacturing systems*, 2003, Vol. 15, p. 37-55.

[25] J.LIU, Y.JIANG, Z.ZHOU: Cyclic Scheduling of a Single Hoist in Extended Electroplating Lines: a Comprehensive Integer Programming Solution. In *IIE Transactions*, 2002, Vol. 34, p. 905-914.

[26] A.MAKHORIN: Modeling Language GNU MathProg. Documentation for GNU Linear Programming Kit, 2005.

[27] P.ŠŮCHA, Z. HANZÁLEK: Cyclic Scheduling of Tasks with Unit Processing Time on Dedicated Sets of Parallel Identical Processors. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Application*, 2007, p. 463-470.

[28] R.JOHNSON: Programmable State-Variable Filter Design for a Feedback Systems Web-Based Laboratory. *Advanced Undergraduate Project Report,* Massachusetts Institute of Technology, 2004.

[29] D.MATĚJÍČEK: Optimalizace Algoritmů pro FPGA. *Diploma Thesis*, CTU in Prague, 2007.

[30] J.A.ROSSITER: Model-Based Predictive Control: A Practical Approach. *CRC Press*, 2003.

[31] E.F.CAMACHO, C.A.BORDONS: Model Predictive Control in the Process Industry. *Springer*, 1997.

Appendix A

A. User Manual

This project is realized in the Matlab environment [http://www.mathworks.com/], more exactly in the version Matlab R2006a. For older versions, there could be some incompatibility, especially in case of creating Simulink model. Created functions can be called from command line or from own m-files. The result of whole project is the Simulink model with automatically generated control function and needed data structures.

A.1. Simulation by Simpler Substitution of TrueTime Library

First necessary step after setting and scheduling the problem is to assign Matlab operations to tasks. These operations will be realized in time due to the final schedule. This code has to be stored in attribute UserParam for each task in the taskset. E.g. for existing task T_1:

```
code = 'in 0';
code = sprintf('%s\n%s\n',code,'x(1) = 5*x(1)-1;');
T1.UserParam = code;
```

This way of adding code for tasks is quite difficult and for larger amount of data also very impractical. Thus, a function *adduserparam* is available. This function adds code from a text file to all tasks in the taskset together. First argument of this function is a taskset object and second argument is a string with the name of the text file. Output object is a taskset with assigned code. Example of use:

```
TS = adduserparam(TS,'data.txt');
```

In the given text file, each task has to be introduced by its name (T_i.*Name*) or by the special character # and then its ordinal number in the taskset, e.g. *#3*. Then it is possible to write down set of commands for an appropriate task. Each set of commands has to be ended by the keyword *endparam*. If there are some commands, which have to be executed at each sample time of the simulation, they have to be closed between keywords *UserParam.begin* and *endparam*. Any text outside the bordered sets of commands, except the commentaries, will cause error. Commentaries are not copied to the control function. Example of text file:

```
UserParam.begin
pause(0.1)
endparam

task1
y = 2*x^2;
z = sin(x);
endparam

#2
x = x+1;
endparam
```

Next step for using simulation is to define inputs and outputs of the Simulink S-Function block, which calls control function in each time sample. Names of inputs and outputs will be then accessible in task commands as variables. For this definition, function *setports* is created. There are two keywords for this function: *Input* and *Output*. First of them introduces part for definition of inputs of the S-Function block and second one for outputs definition. After each of them, next argument is the name of the input/output and then its size (vector length). If only inputs (or outputs) are needed, only one keyword can be used. Output of this function is a structure that is one of the input arguments for the main function. Example how to create two inputs with size 1 and one output with size 2:

```
ports = setports('Input','w',1,'e',1,'Output','control',2);
```

Now is possible to call the main function of the project – *taskset2simulink*. This function will generate Simulink model and control function and other data structures from the given taskset with time schedule. Syntax of the function call:

taskset2simulink(file, TS, ports, VRin, stopTime, varargin)

Description of parameters:

file	- name of virtual reality file (has to be ended by postfix *.wrl*); all created functions and files will contain this name.
	- if virtual reality is not needed, arbitrary name of project can be set.
	- it is possible to set empty argument *[]* and project will be named *project1*.
TS	- taskset object.
ports	- structure with information about inputs and outputs of the control block.
VRin	- structure with information about inputs of the virtual reality block .
	- empty argument *[]* if virtual reality is not used.
stopTime	- stop time of the simulation.
varargin	- additional information; set in format: (‘*Property name*‘, *Property value*).

- ‘*Sample*‘	– value of sample time; default value is one.
- ‘*Period*‘	– value of schedule period; without repetition as default.
- ‘*Simulink*‘	– string with information about Simulink model generation.
	– ‘*off* ‘ if Simulink model is not needed to be created.

Example of the main function call for the case without virtual reality, name of project will be *dsvf*, sample time one second, stop time 50 seconds, period of schedule 10 seconds and we do not need to generate Simulink model:

```
taskset2simulink('dsvf',TS,ports,[],50,'Period',10,'Simulink','off')
```

A.2. Visualization with User-defined Virtual Reality

Technique for the visualization of the scheduling results is the same as for the simulation with one additional step. It is necessary to define inputs for the virtual reality block in Simulink. For this purpose, function *VRcontrol* is available. Input arguments are the pairs of strings, where first one is the exact name of the object in the virtual reality and second one is the property, which we want to refer to. Output of this function is the structure with given information. Example:

```
VRin = VRcontrol('Arm1','translation','ColorMachine1','diffuseColor');
```

Now, first input of the virtual reality block will refer to the object called *Arm1* and it will be possible to change the object property *translation* via this input.

If we want to control some inputs of the VR block by outputs of the control block, it is necessary to have these outputs/inputs at the same ports of blocks. For example, if we want to control the third input of the VR block, an appropriate output of the control block has to be also on the third position. In the situation when more outputs from the control block than inputs to the VR block are needed, these outputs have to be defined after definition of corresponding outputs/inputs. For example, if we want to connect first two outputs of the control block with the VR block and two other additional outputs are needed, the function calls are following:

```
ports = setports('Output','controlArm',3,'signalColor',3,'U',1,'out',2);
VRin = VRcontrol ('Arm1','translation','ColorMachine1','diffuseColor');
```

It will be possible to control translation of the object *Arm1* from the control block output *controlArm*. It is the same for the output *signalColor* and the property *diffuseColor* of the object *ColorMachine1*. By calling main function with these structures together with defined virtual reality project, we get Simulink model shown on Figure.

```
taskset2simulink('my_poject.wrl',TS,ports,VRin,500);
```

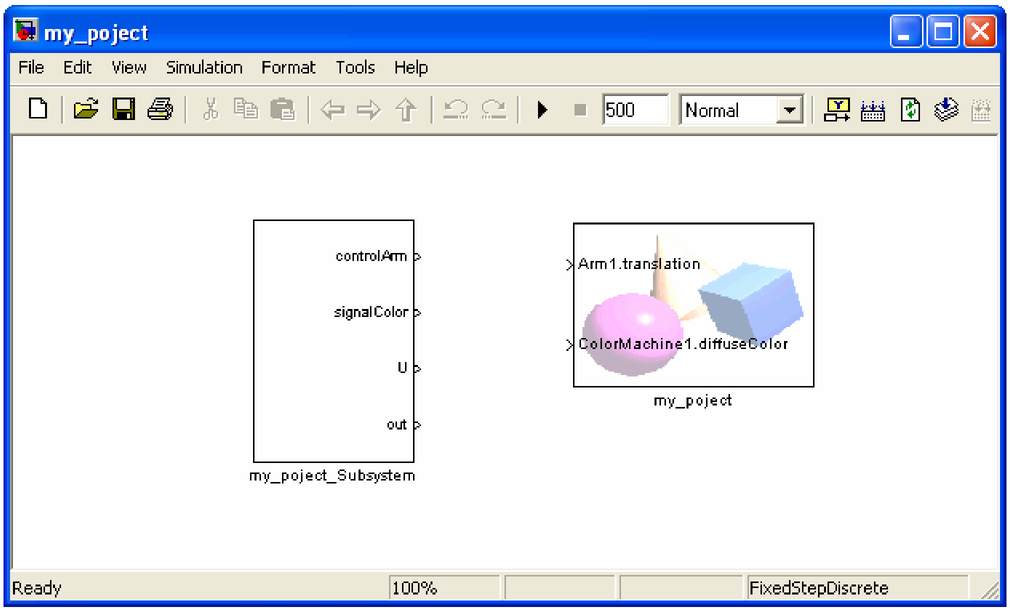


Figure A.1 - Generated Simulink model

A.3. Definition of Commands for Tasks

There are three ways how to define commands for tasks by relative time:

1) By the keyword *repeat*, followed by three numbers separated by colons – *start*, *step* and *stop*. First number determines the start time for executing of the following block of commands. This time data determines time moment to start of executing these commands referring to the beginning of a task in the schedule. Second number determines time space between two repetitions of this block of commands and the last number determines the end of executing commands referring to the beginning of a task in the schedule. Time data *start* and *stop* can overreach the borders of a task in the schedule, i.e. *start* can be negative and *stop* can be greater than processing time of a task. Therefore, it is possible to define some operations before and after of a task in the schedule. Time data *step* has to be positive number.

2) By the keyword *divide*, followed also by three numbers separated by colons – *start*, *step* and *stop*. Meaning of these time data are very similar to the previous case, with only one difference: data *start* and *stop* are not in real time units. Instead of this, they are proportionally related to the processing time of a task. Generally, they are decimal numbers and it is also possible to execute some commands outside of a task in the schedule.

3) By the keyword *in*, followed by only one time data, which determines in what time related to beginning of a task in the schedule will be the block of commands executed.

Code for one task can be for example following:

```
task1
repeat 0:1:7
T1trans(1) = T1trans(1)+1;
in 7
ColM1 = [1 0 0];
divide 0.5:1:1
Drill1(2) = Drill1(2)-0.04;
```

```
repeat 19:1:29
Drill1(2) = Drill1(2)+0.04;
in 29
ColM1 = [0 1 0];
repeat 29:1:35
T1trans(1) = T1trans(1)+1;
endparam
```

Until a new keyword is used, all commands belong to the previous keyword. It is possible to use all standard Matlab functions and in addition, function *getVRpar* is available. Via this function, it is possible to obtain value of any accessible property defined in the virtual reality file. First argument of this function is the name of VR file, second argument is a string with name of the object in VR and the last argument is the name of property that we want to get. Output value is a vector with relevant size. Example of use:

```
value = getVRpar('my_world.wrl','Earth','rotation');
```

To store any numerical value between two time samples, there are internal states *x(i)*. Main function will automatically define needed amount of states according to the highest used index. To refer states, use format *x(3)*, it is not possible to use notation *x(1:3)*. It is also possible to refer to the inputs and outputs defined by the function *setports* with the same names as in definition.