Scheduling Drilling Processes With Petri Nets

Nejm Saadallah

Thesis submitted in partial fulfillment of the requirements for the degree
of Philosophiae Doctor (Ph.D.)
August 12, 2013

ISBN:978-82-7644-542-8
PhD: Dr avh. Nr 201
ISSN: 1890-1387

Dedicated to Nadia, Teo and Daria

Abstract

Safety issues in drilling are related to two facts: Wells are becoming more complex, and manually piloting a drilling rig is a difficult task which requires highly skilled personnel. Consequently, improving safety is conditioned on a better anticipation of the well behaviour, and an easier way of operating drilling rigs.

On top of safety issues comes the drilling industry vision of autonomous drilling control systems. This vision aims at realizing a drilling control system, which not only is capable of executing a drilling program but can also automatically respond to incidents. However, we need to overcome a number of challenges before the autonomous drilling vision comes true.

In this thesis we aim to address the following challenges: First, we need to provide a system component which guaranties a safe control of the rig. That is, any control operation that can be performed has to be legal. Such a component is called *control supervisor*.

Second, we need to provide a capability for handling incidents. This includes processes that can monitor the well dynamics and trigger actions to cope with eventual incidents. Such processes are called *reactive processes*.

Third, because *reactive processes* could trigger conflicting actions, we need a mechanism to coordinate them. We call such a mechanism *reactive process scheduler*.

Realizing a *control supervisor* has its foundation in the Discrete Event System (DES) paradigm. The main problem we address in that context is to provide a DES model that captures the dynamics of the rig, and which can be checked for correctness.

Realizing a *reactive process scheduler* is related to obtaining an

emergent behaviour out of basic ones. A basic behaviour is associated to every *reactive process*, and the task of a *reactive process scheduler* is to coordinate those *reactive processes* in order to obtain a satisfactory behaviour of the overall system.

The main contributions of this thesis are:

1. Bringing to light some hidden challenges related to drilling control systems.
2. Including two system components to the existing drilling control system architecture: A *control supervisor*, and a *reactive process scheduler*.
3. A Petri net class to model the *control supervisor* which properties can be fully analysed.
4. A theoretical approach for modelling *reactive processes* and their scheduling.

All in all, this thesis aims to not only ease the development of safer drilling systems, but also to take a step towards the more ambitious vision of autonomous drilling.

Preface

This thesis is submitted in partial fulfilment for the degree of Philosophiae Doctor (Ph.D.) at the University of Stavanger (UiS). The research has been carried out at the Department of Electrical Engineering and Computer Science, and the International Research Institute of Stavanger (IRIS).

This research was funded by the joint industry project AutoConRig, involving the following participants: National Oilwell Varco (NOV), Statoil, Baker Hughes, Computas AS, Det Norske Veritas (DNV), International Research Institue of Stavanger (IRIS), University of Stavanger (UiS) and University of Oslo (UiO).

Readership

This thesis aims at improving existing drilling control systems using knowledge and theory from the Discrete Event System (DES) field in general and Petri nets in particular. Even if the thesis introduces most of the used concepts, its full appreciation requires a good background in DES formal methods, and some knowledge of drilling.

Acknowledgements

I would like to take this opportunity to thank a number of people who have helped me completing this thesis.

First of all, I would like to thank my thesis advisor Professor Hein Meling. Without your guidance this thesis would not have been possible. Thank you for being generous with your time and for sharing your knowledge and experience with me.

Thanks to Professor Reggie Davidrajuh, for being at the same time a tough, and a kind supervisor. I owe you a big thank for directing me into the wonderful domain of Discrete Event Systems.

Thanks to my friend and colleague Dr.Benoit Daireaux for accepting to supervise this thesis despite an already overloaded every day. Your advices have truly influenced this thesis, I thank you for that.

I would like to thank Associate Professor Slawomir Samulej, Professor Lars Kristensen and Associate Professor Erlend Tøssebro for taking the time to serve on my dissertation committee.

Thanks to Eric Cayeux for his invaluable advices, and for sharing his knowledge with me. Thanks to the Drilling and Well Modelling group leader Helga Gjeraldstveit for her encouragements. Thanks to all may colleagues at IRIS and UiS for contributing to a joyful working environment.

Thanks to all the project participants of AutoConRig: National Oilwell Varco, Statoil, Baker Hughes, Det Norske Veritas, Computas AS, IRIS, UiS and UiO. In particular, I wish to thank Henning Jansen for including this PhD into the AutoConrig project, Jens Ingvald Ornaes for following my progress, Professor Roar Fjellheim and Professor Chunming Rong for all their advices.

Thanks to Elisabeth Fiskå from the Department of Electrical

Engineering and Computer Science for all kinds of administrative help.

I would like to thank all my friends and family members who supported me in any respect during this period.

Finally, and most of all thanks to my lovely wife Dasha, without whom I would have neither begun nor finished this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Symbols And Abbreviations

$\bullet p$	pre-transition of p
$\exists$	there exists
$\forall$	for all
$\leq$	$m \leq m'$ stands for m' covers m
$\mathbb{N}$	the set of Natural numbers
ω	very large number
σ	Sequence of transitions
${}^\bullet t$	pre-place of t
${}^\circ p$	set of inhibited transitions by p
${}^\circ t$	set of inhibiting places of t
$a \in B$	element a in set B
$A \subset B$	A is subset of B
CCS	Calculus of Communicating Systems
CES	Set of circular elementary structures
ces	circular elementary structure
CPN	Coloured Petri nets
$CPTI$	Cohesive P/T nets with inhibitors

CSP Communicating Sequential Processes

DES Discrete Event System

E Set of Edges

ES Set of elementary structures

es elementary structure

FES Set of flat elementary structures

fes flat elementary structure

$FIFO$ fist in first out

g_i goal i

G_m set of goals satisfied by m

I set of inhibitor arcs

m Petri net marking

m_0 initial marking

m_i Petri net marking i

M_ω Set of extended markings

MEX mutual exclusive relation

Mg_i set of markings satisfying g_i

$MICPTI$ Mutually Inhibited Cohesive P/T nets with inhibitors

$MINC$ mutual inclusive relation

P the set of Petri net places

P/T Place Transition nets

$p^\bullet$ numbers of tokens of p at making m

$p^\bullet$ post-transition of p

p_i Petri net place i

$PINC$ partial inclusive relation

$proc_i$ process i

PT the name of the quadruple defining P/T nets

PTI Place Transition net extended with inhibitor arcs

$R(m)$ reachable markings from m

SFC Sequential Functional Chart

T Petri net transition set

$t^\bullet$ post-place of t

t_i Petri net transition i

$TINC$ partial exclusive relation

V Set of Vertices

Part I

Overview of Research

Chapter 1

Introduction

Drilling technology has seen great advances over the past two decades, and as the world's energy demand continues to grow, large strides will be made for further advancement. Drilling complicated wells with lengths beyond 7 km have become common practice [17]; something that were unthinkable only a few years ago. Finalized in May 2008, the BD-04A well measures 12.3 km long with a 10.9 km horizontal section, and placed in an oil reservoir spanning only six meters [29].

Nevertheless, drilling a well remains dangerous and costly. For example, drilling a well in the North Sea may take up to 60 days with a typical rig rental of 500.000 dollars per day [125]. Hence, a major contributor to the costs of realizing a well is the daily rig rental. Thus to reduce costs, the obvious target is to reduce the non-productive time. It is even more important to reduce the risk of incidents as these tend to increase with the well's complexity.

Even if cost and safety are major challenges, the drilling industry remains optimistic about the future, because overcoming these challenges enables a broader application of drilling. Among these applications, the geothermal energy is the most promising [127], because drilling deeper at lower costs implies higher energy conversion efficiency [82]. Second, the remaining oil and gas resources are found in areas with rough and stormy weather, which implies a higher risk of incidents. Third, the developing CO_2 storage technology also implies drilling a well [55].

These drilling applications are pushing the industry towards the

ambitious vision of *autonomous drilling*. We define *autonomous drilling* as the ability of a drilling control system to run a drilling plan without damaging the rig or causing any hazardous well incidents.

In state-of-the-art drilling systems, the drilling process is mainly handled by a driller that operates the rig machinery through a drilling control station. The rig machinery is in turn composed of a multitude of devices that are largely operated independently by the driller. Hence, coordination between the different devices is generally entrusted to the driller's abilities and experience. Moreover, the driller performs manoeuvres based on input from sensory observations. Interpreting these observations can be very intricate, making it difficult to identify the proper next manoeuvre, and may eventually cause an incident [73, 76, 42]. In this thesis we consider two types: *well incidents* and *machine control incidents*.

A *well incident* is usually caused by applying wrong drilling parameters, such as a too high flow-rate, or a too high drill-string velocity, which can, not only fracture the formation but also cause a *gas kick*. A *machine control incident* occurs when the rig is not correctly operated. For example, releasing the drill-string, before activating the *power-slips* (used to suspend the drill-string), causes the drill-string fall into the well.

Safety in drilling refers to *machine control safety* and *well safety* [26], where the first aims to avoid *machine control incidents*, and the second to avoid *well incidents*. In order to improve *machine control safety*, piloting a rig has to become easier. As for improving *well safety*, we need to anticipate well incidents, and trigger appropriate responses to handle them.

We consider that guaranteeing a safe machine control, and enabling automatic responses to well incidents are necessary conditions for achieving the *autonomous drilling* vision. These conditions will be addressed in this thesis.

1.1 Motivation

Traditionally, the main challenge of the drilling discipline is to realise long reaching wells. Long wells, especially those with long horizontal

sections are difficult to drill, because they impose significant mechanical and hydraulic constraints. In the planning phase, drilling engineers must answer whether the target depth can be reached, and how to reach it. For that, they need physical models, software tools using those models, and sufficiently powerful drilling rigs.

Today, in addition to reaching the target depth, planning a well requires costs analysis, safety analysis and alternative plans.

The result of the planning phase is a *drilling plan* for the drilling crew to use. A *drilling plan* includes information about the characteristics of the *drilling fluid*, the planned trajectory of the well, the casing sections, the type of drill-string, drilling parameter under different conditions etc.

When drilling a well, the driller executes the *drilling plan*, and eventually reach the target depth. Because a *drilling plan* cannot account for all possible situations, the drilling crew is supported by a monitoring team of engineers whose main task is to estimate the well conditions and suggest actions in response to changing well conditions. Typically, when the friction in the well increases, the driller should adjust the flow rate or the rotational speed based on inputs from the monitoring team. About ten years ago, these teams were located on the rig, but today they usually reside in *Drilling Operation Centres*. These centres run 24/7, and monitor ongoing drilling operations in real-time using dedicated software tools called *Drilling Decision Support Systems* [41, 42].

Some incidents require immediate actions, hence, the driller can not always rely on the monitoring team. This issue has motivated the development of drilling control systems that limit the drilling parameters available to the driller based on the well conditions. These are usually referred to as *drilling-by-wire-systems* (or *safe-guarding-systems*). The idea behind *drilling-by-wire-systems* is to enforce smooth drilling operations avoiding well incidents, and some of these systems are already deployed offshore [76, 77, 41, 42, 50, 115].

The challenges of the *drilling-by-wire-systems* are mainly related to their well models. Improving those models results in a high confidence when limiting drilling parameters such as rotational and axial velocities. However, *drilling-by-wire-systems* do not implement

automated responses to *well incidents*. That is, the rig control is still left to the driller.

From the innovation of *drilling-by-wire-systems* and *Decision Support Systems* emerged drilling simulators [43]. The objective of these is to provide a realistic drilling environment and generate simulated well incidents in order to train drillers on a rapid assessment and reactions to dangerous situations.

Despite the good results from *drilling-by-wire-systems*, there is hesitation in the industry to take one step further and implement automated responses. For example, if a situation that requires an immediate activation of the mud pump occurs, today's systems can in the best case generate alarms, while it is up to the driller to respond to those alarms or to disregard them. The reason for this hesitation is that there is no known approach that can automatically generate consequent actions for well incidents without compromising *machine safety*. So, before taking this step, *machine safety* must be guaranteed. Thus, when taking a step towards *autonomous drilling*, *machine safety* and *well safety* become even more important.

Several research and industrial initiatives are working towards the *autonomous drilling* vision. One such initiative is the Continuous Motion Rig (CMR) [91, 90], which aims at reducing drilling operation time, and providing better well stability. CMRs differ from existing rigs in the way they handle pipe connections. When running the drill-string down the well on a conventional rig, the downward motion needs to be stopped, the drill-string needs to be suspended using the power-slips, and the mud pump needs to be turned off. These combined can have undesired side effects on the conditions in the well. In contrast to conventional rigs, CMRs aims to enable attaching new pipe segments while maintaining the downward motion, which is also combined with continuous fluid circulation when needed. That is, there is no need to activate power-slips or to turn off the mud pump. As a result, operations are expected to run smoothly and continuously without interruptions. This leads to faster drilling and improved well integrity.

The Seabed Rig project [89, 8] is another initiative towards *autonomous drilling*. The objective of the Seabed Rig project is the construction of a new generation drilling rig that is placed on

the sea-bed and operated from an offshore support vessel. The rig will be equipped with appropriate cameras, and sensors for providing sufficient situation awareness to the driller. Because the rig will be placed on the sea-bed, new and more robust drilling equipment must be developed. Since human interventions will be difficult, such a system must have the ability to handle incidents.

Another initiative towards the autonomous drilling vision is the AutoConRig project [104, 105], which is a joint industry project with the following participants: National Oilwell Varco (NOV), Statoil, Baker Hughes, Computas AS, Det Norske Veritas (DNV), International Research Institue of Stavanger (IRIS), University of Stavanger (UiS) and University of Oslo (UiO). The AutoConRig project has two main objectives [105]:

1. *The primary objective of this project is to analyze, develop and test an autonomous and semi-automated drilling control system for Oil and Gas Drilling in High North areas, where unmanned drilling rigs placed on the sea bottom can be used to eliminate constraints from extreme conditions. The outcome of the main objective will be used to demonstrate an automated tripping sequence where predictive control parameters from an advanced well model is executed by an autonomous control system. The automated tripping sequence takes into account characteristics and constraints in the well, avoiding damage to the well and at the same time optimizing the tripping sequence.*

2. *The secondary objective for the project is to standardize communications protocols for drilling control systems and a framework for advanced software agents, which is a prerequisite to fulfil the integration scope of the primary objective.*

This thesis is part of the AutoConRig project, and addresses modelling aspects that must be considered in order to obtain such an automated tripping sequence. As for the role of software agents, this thesis proposes a definition of their action domain. However, we shall use the term reactive process rather than agent.

This thesis addresses safety from two distinct, yet interrelated aspects. One focuses on *machine control safety* by keeping the rig

dynamics within the set of acceptable states. The other focuses on *well safety* by keeping the well dynamics within the set of acceptable states.

Obtaining *machine control safety* requires a model that captures the rig dynamics. We abstract the rig dynamics to the domain of Discrete Event Systems, because they offer suitable model checking capabilities.

To obtain *well safety*, we propose an approach that aims at achieving satisfactory emergent behaviour out of basic processes. We call those basic processes for *reactive processes*, where their role is to observe key aspects of the well dynamics, and trigger actions using the rig.

1.2 Objectives

The vision of this thesis is to propose an approach for executing a drilling plan without violating the rig legal behaviour or damaging the well. To move towards that vision, we define three main objectives: Safe Machine Control, Safe Well Control, and Automated Plan Execution.

1.2.1 Safe Machine Control

The first objective of this thesis is to propose a method for modelling drilling control systems that guarantees a safe control of the rig machinery. For that, we need to find an appropriate domain of abstraction that best fits our needs. This issue is discussed in Chapter 4 which can be summarised in: Discrete Event Systems (DES) [25] and their modelling by means of Petri nets [108].

Modelling DES with a Petri net formalism has the following attractive characteristics:

1. Petri nets provide an explicit model for the system dynamics.

2. Many properties of the modelled system can be validated using model checking [107, 99].

3. Petri nets can provide a means for systematically determining sequences of actions.

The first reason is always true, the two others are true for some classes of Petri nets only. That is, not all Petri net models can be checked for behavioural properties. Because we are faced with such issues when attempting to capture the rig dynamics in a Petri net model, we propose a specific class that matches our needs. This class is described in Chapter 6.

1.2.2 Safe Well Control

The second objective of this thesis is to enable *reactive processes* to trigger actions for handling well incidents. Capturing the physical behaviours of the well is out of the scope of this thesis. However, combining partial observations of the well with appropriate responses can still lead to a satisfactory global behaviour.

This second objective assumes that a safe machine control is obtained (the first objective). It particularly assumes that we have a method for finding sequences of actions between a source and target state. Under this assumption, the problem becomes to model reactive processes such that they do not conflict with each other. That is, given that a process has triggered, can another one also trigger without stopping the first one? Answering this question provides a foundation for designing a scheduler for reactive processes. This issue is addressed in Chapter 7.

1.2.3 Plan Execution

The third objective of this thesis is to propose an approach for an automated execution of a drilling plan. This objective builds on the two previous ones, and attempts to stretch towards our vision.

To reach this objective we assume a safe machine control, and a set of reactive processes capable of handling well incidents if they occur. On top of those two concepts we propose an approach for executing segments of drilling plans.

1.3 Contributions

A Formalisation of Drilling Control Safety

Existing drilling control systems are usually presented at a high level, providing little or no view into fundamental control system problems. This thesis formalises key issues related to drilling, and points at fundamental aspects that need to be addressed before we can move towards autonomous drilling. We emphasise two facts:

1. The well exhibits a complex behaviour which is in the best case estimated through partial observations. This means, that a precise control of the well dynamics is hard or even impossible to obtain.
2. The rig is the tool by which the well is drilled and by which drilling related problems are solved. This means that the rig must be operated properly and that the rig dynamics must be kept within the legal states.

In Chapter 3 we present issues related to drilling, and propose an improvement to existing drilling control systems based on the above mentioned facts. In Chapter 4 we present arguments for using Petri nets as a modelling formalism.

A Subclass of Petri nets with Inhibitors

When modelling the dynamics of the rig in Petri nets, we found it necessary to use inhibitor arcs. However, Petri nets with inhibitors are problematic, because they are hard to analyse. As a contribution to the field of Discrete Event Systems and Petri nets, we propose a sub-class of Petri nets with inhibitors which can be fully analysed. This class of Petri nets is presented in Chapter 6 and later on used to model the rig dynamics in Chapter 9.

Reactive Processes on top of Petri Nets

Because the well behaviour is hard to capture in a precise model, we use a strategy that uses partial observations in order to trigger actions. In other words, we want to obtain a satisfactory behaviour of the well using basic reactions to observations. In this contribution

we propose a method for modelling reactive processes on top of Petri nets. This method will be used to provide a process scheduler that is capable of determining which reactive processes can take actions and which have to wait (Chapter 7).

A Petri Net Model of a Drilling Control System

To demonstrate the usefulness of the above mentioned approaches, we present a Petri net model of a rig, and a set of reactive processes on top of that model. This is done is Chapter 9, where we first show how the properties of the rig dynamics can be analysed. Second, we show that we can obtain a satisfactory behaviour by combining an appropriate set of reactive processes. Finally, we suggest an approach for executing a plan and systematically handle incidents when they occur.

1.4 Outline

This chapter has presented the motivation of this thesis, its objectives and contributions.

Chapter 2 starts with an introduction to drilling, and drilling related issues. It also gives an overview of existing drilling control systems. Chapter 3 presents the limitations of existing drilling control systems, and suggests improvements.

Chapter 4 gives an introduction to the different theoretical domains of interest. It gives a short introduction to DES, Supervisory Control and Reactive Systems, and relates them to our problem of modelling the rig dynamics. This Chapter also explains our choice of using Petri nets as modelling formalism for the rig control. Finally, this chapter explains the concept of emergent behaviour.

Chapter 5 presents Place/Transition nets (P/T), and mainly focuses on its analysis by means of state space exploration methods. That is, using either a reachability graph for nets representing finite systems, or coverability graph for nets representing infinite systems. The objective of this chapter is to present the necessary definitions, the benefits and limitations of P/T nets. This chapter can be viewed

as background material for Chapter 6.

Chapter 6 presents our main theoretical contributions. It introduces a new class of Petri nets extended with inhibitor arcs which has interesting properties. In particular, this class of nets can model some infinite systems, and can also model situations that P/T nets fail to model. Another benefit of this class of Petri nets is that it can be analysed for almost all the properties of interest.

Chapter 7 presents the concept of *reactive processes*. It gives them a formal definition, shows how they can be modelled on top of Petri nets. We then present a method for analysing the interaction between reactive processes and show how a process scheduler can take advantage of that analyses.

Chapter 8 presents our newly developed software package based on the results from Chapter 7 and 6.

Chapter 9 presents a drilling control model that uses the above mentioned concepts. It shows how our Petri net class can be used to model the dynamics of the rig, and how the properties of that model can be analysed to derive conclusions about the correctness of the system. We also show how to use reactive processes to obtain a satisfactory emergent behaviour of the overall system. Finally, this chapter extends the use of reactive processes to represent an *execution plan*. All-in-all this chapter demonstrates that under some assumptions, our method can run an operation plan, cope with incidents as they occur, and without violating control-specific constraints.

Chapter 2

Introduction to Drilling

In this chapter we start by giving a short introduction to the physics of drilling, how things usually work, and what should be avoided. We also discuss existing systems role in addressing today's drilling problems.

2.1 The Basics of Drilling

We start with a short and informal introduction to drilling. Covering the complete spectrum of the drilling field is not our goal. However, we recommend the following [9, 35, 11]for the interested reader.

A drilling rig is a structure specially built for drilling wells. It is composed of different devices, each of them designed to perform a specific task. Figure 2.1 presents a schematic of the different rig devices. The crow block (13 in the figure) is at about the hight of the Derrick(14). The travelling block (11) is pulled up, and lower down using the Drill line(12), which in turn is attached to the Draw-works. The to drive (18) is connected to the drill-string (25) and attached to the travelling block. The standpipe (8), the kelly-hose (9) and the Goose-neck (10) constitute a flexible pipe, starting from the mud pump (4) and ending at the top-drive. From an abstract perspective, a drilling rig has three degrees of freedom, reflected in three subsystems: rotation system, hoisting system, and fluid circulation system.

Drilling a well consists of putting the drill-bit in the earth, press

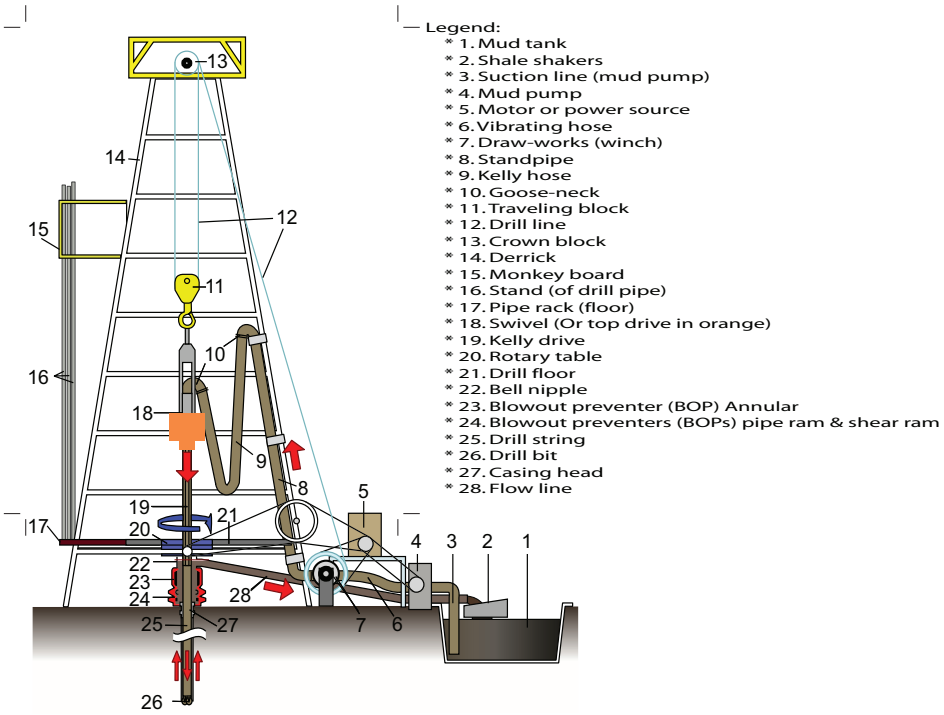


Figure 2.1: A simple illustration of an oil drilling rig. The figure is obtained from [137] and is licensed under Creative commons [28]

it against the ground and rotate it. As the drill-bit smashes the rock into cuttings, these must be removed continuously while rotating the drill-bit. To move cuttings away from the bit, a special liquid called drilling fluid (also called drilling mud) is pumped through the drill-string into to the drill-bit to push away the cuttings and carry them back to the surface. The drilling fluid flows through the drill-bit via the bit nozzles (holes with small diameter), the fluid jets out, and lifts the cuttings from the bottom of the well. The cuttings and the fluid flow back to the surface via the annulus, which is the space between the drill-string outer wall and the wall of the hole. Once at the surface the drilling fluid is treated, discharged to the suction tank, and pumped back down the drill-string to the well.

Another important role of circulating a drilling fluid is to keep the well pressure between the fracturing and collapse pressure. That is,

the pressure inside the well should be less than the well fracturing pressure, and more than the well collapse pressure. Because the drilling fluid is the means by which the pressure in the well is controlled, its properties must be carefully chosen.

The mud pump is connected to a rotating device called top-drive via a rotary hose. The top-drive is responsible of rotating the drill-string, and thus must have powerful built-in motors to obtain the required torque to drill the rock. The top-drive is in turn suspended to the travelling block which is elevated or lowered using the hoisting system. In order to put pressure on the drill-bit the hoisting system has to be lowered just enough to push to drill-bit, but not too much to preventing the drill-string from buckling.

Fluid circulation, rotation and hoisting are what constitute the basic mechanisms for not only drilling a well, but also for handling well incidents. Examples of well incidents are: gas kick, stuck pipe, pack off, hole collapse, formation fracturing etc. When a well incident occurs, fast responses are usually required. Typically in form of adjustments to flow-rate, rotational or axial velocity.

When drilling through a zone containing gas with a too light drilling fluid, the gas can enter the well and migrate to the surface, causing a so-called gas kick. A possible response to such a case is to pump the fluid with sufficiently high flow-rate to increase the well pressure and eventually push back the gas to the formation. However, using a heavier mud, we run the risk of fracturing the formation causing fluid losses. That is, the pumped fluid does not return back to the surface, but is pushed into the formation surrounding the drill-string. This situation is very undesirable because it can cause a hole collapse, which would not only suspend the drill-string, but also cause an uncontrolled kick.

An uncontrolled kick is also called a *Blow out*, and its occurrence requires the activation of the Blow Out Preventer (BOP). The BOP is a mechanical device capable of locking the well in order to contain the kick. After the BOP is activated, the well is usually lost.

Other tools are also used in the drilling process, and are often seen as utilities responsible for intermediate operations. For example, connecting a new drill-pipe to the drill-string would require the activation of the power slips, which is used to suspend the drill-string

and preventing it from falling into the well. Another tool is the iron-roughneck which is responsible for applying the make-up or break-off torque when connecting or disconnecting drill-pipes. There is also a special valve called Internal BOP (IBOP), which is used to prevent drilling fluid from falling on the drill floor. Typically, the Internal BOP must be closed under pipe connection and opened when the fluid is being pumped into the well.

2.2 High-Level Drilling Operations

In the drilling domain terminology, drilling operations are often described at a high level; the most common ones will be described in this section.

Tripping-in and out

Tripping-in is the process of running the drill-string down to the bottom of the well. It consists of lowering the drill-string until a certain distance from the drill floor (about 1 meter above the drill-floor).The power slips are then activated to keep the drill-string in suspension (avoiding its fall into the well). After that the top-drive is disconnected from the drill-string, and elevated to a certain distance above the drill floor (usually about 31 meters). A new stand (3 connected drill-pipes) is then brought to the well center in order to connect it to the drill-string. Connecting a stand to drill-string is done using the Iron rough neck which responsible for applying the necessary make-up torque. The top-drive is then connected to the drill-string.

This process is repeated until the drill-bit reaches the bottom of the well. On the other hand, tripping-out consists of pulling the drill-string out of the well in a reversed process.

Reaming and Back-reaming

Reaming and back-reaming aim at smoothing the hole. These are usually performed after a stand (three connected pipes) has been

drilled, because a newly drilled hole is not necessarily as smooth as it should be. Reaming consists of slowly lowering, rotating and circulating fluid through the drill-string. While back-reaming consists of the same operations, but with the drill-string being pulled out rather than lowered into the well.

The Drilling Operation

Drilling as an operation can start when the drill-bit has reached the bottom hole. The key parameters here are the weight on the drill-bit, the rotational velocity and fluid flow-rate. These parameters have to be tuned depending on the well depth, the drilling fluid in use, the type of rock being drilled in, and of course the geo-pressure prognosis (fracturing and collapse pressures). The progress in drilling is reported in a parameter called rate of penetration (ROP).

Friction Testing

A good estimation of the well friction is important when determining the well conditions. A too high friction may indicate that cuttings are being accumulated in the well, which will not only require more torque to rotate the drill-string, but also increases the risk of fracturing the well. A friction test is usually done after reaming, and consists of pulling the drill-string, followed by a rotation of the drill-bit when it is slightly above the bottom hole.

Hole Cleaning

A high well friction is often due to an accumulation of cuttings in the well. Circulating these cuttings out is done by applying an appropriate flow-rate. It is sometimes necessary to change to another fluid with different properties, e.g one that has better cutting transport capability.

Reciprocating

Leaving the drill-string without any motion can lead to a so called differential sticking. Differential sticking occurs when a relatively large drill string surface stays in contact with the formation, and that the pressure in the well is higher than the formation pressure. For this reason the drill-string needs to be kept in motion to avoid a long contact with the well wall. Reciprocation is usually done by applying an upward and downward motion of about 5 meters combined with a drill-string rotation with low frequency (about 60 rpm).

Surveys Receiving

Slightly above the drill-bit, there is a special tool called Measurement While Drilling (MWD). MWD measurements concern down hole parameters only, such as directional information, formation evaluation, down-hole pressure and temperature.

At regular intervals, MWD data is sent from the bottom hole to the surface. The data is sent using mud pulses that are generated by the MWD tools, which in turn are decoded at the surface. The rate at which these data is sent depends on the equipment, and variates from 20 to 1.5 bps [135].

In contrast to MWD data, surface data concern measurements taken at the surface, and are thus available at a much higher bandwidth at Mbps scales.

Other Considerations

Above we have introduced what we consider to be the most common drilling operations. It is clear that these operations are performed using different sequences of basic operations. In fact depending on the needs, one could include other high-level operations by describing their corresponding sequences. However, what we would like the reader to retain from this section is that, based on the conditions of the drilling site, the well, or user's needs, sequences of basic operations must be performed. One of the issues treated in this thesis is an approach for expressing high-level operations using basic ones.

2.3 Drilling and Safety

There is no doubt that the wells that need to be drilled will be more and more challenging. One of the fundamental questions that the drilling industry has to answer is how to deal with well complexity without compromising safety?

According to a study conducted by Petroleum Safety Authority of Norway on the causes of kick incidents [109, 102], up to 15% of the kicks where caused by human errors, and 13% were due to poor detection. The report also points to other causes, such as poor well design, equipment failure, or organisational issues. However, these causes are not relevant for our work and will not be discussed further.

Machine Incidents

Machine incidents [73, 26] refer to those incidents caused by a system dysfunction or human mistakes when operating the rig. Here are some examples:

1. When connecting a new pipe to the drill-string, the power slips needs to be activated in order to prevent the drill-string from falling into the well. So, if the power slips are released before the top-drive is connected, the drill-string will fall.
2. Collision with the crown block when pulling the drill-string, which can cause objects to fall on the drill-floor.
3. Starting the mud pump before the top-drive is connected, or when the IBOP is open, will spill mud on the drill-floor and may be dangerous for the drilling crew.
4. Rotating the drill-string when the power slips are activated, or activating the power slips when rotating may cause serious damages.

According to [73, 26] machine control related incidents are generally due to lost concentration, or poor communication between the drilling crew members. To reduce these types of incidents one could require highly skilled drilling crews, but in this thesis we aim to show that machine related incidents can be avoided by the control system.

Well Incidents

Well incidents [73, 26] are those undesired events which are related to the well conditions. It is difficult to obtain a precise map of the actions that could cause particular incidents, or which incidents are followed by others. Nevertheless, we list some of the common well incidents and their believed causes below:

1. Stuck pipe: This incident is probably the most common under drilling operations. The drill-string is considered stuck when it is no longer possible to rotate nor to elevate it. The drill-string can get stuck for different reasons such as differential sticking, poor hole cleaning, complex well trajectory etc.
2. Fracturing: This incident happens when the fluid pressure exceeds the fracturing pressure. Fracturing the well has the immediate consequence of fluid losses, i.e. fluid is pushed into the rock. Fracturing the well can also cause a kick.
3. Hole collapse: This incident can be seen as the reverse process of formation fracturing. A collapse happens when the fluid pressure is less than the collapsing pressure. In this case, the rock will fall into the well, usually causing a stuck-pipe. Collapsing the well means that fluid can not be circulated any more, and the risk of a kick increases dramatically as it becomes harder to control the kick without fluid circulation.
4. Kick: This incident happens when fluids; gas or oil enter the well and migrate to the surface. Handling a kick depends on the volume of the influx, the drilling fluid in use, and also the surface equipment. A common response is to increase the drilling fluid rate or its weight in order to increase the well pressure and eventually control the influx. If the influx is too important, some rigs are equipped with a so called flare. In that case, the gas is canalised from the annulus via the annular chock up to the flare for burning. However as mentioned earlier, the last barrier for a kick is the BOP and its failure could lead to catastrophic scenarios.
5. Pack-off: This incidents happens when the drill-string is not totally stuck but can not be totally pulled or rotated. It can be due to a small well collapse, accumulation of cuttings, or

simply an object that has fallen into the well. If a pack-off is not handled correctly it can cause a formation fracturing or a stuck-pipe.

It is difficult to determine exactly the causes of the above mentioned events. Drilling teams usually use their experience and specialised software tools in order to asses the well conditions, identify critical situation, and eventually come up with remedial actions.

Obtaining a precise control of the well dynamics requires not only precise physical well models, but also that these models are fast and precise in their computations. Starting from sensory observations, one would like to estimate the well states that actually reflect the observations. In this thesis we assume that the well dynamic is partially observed by sensory data, but we do not assume that sensory data reflect the complete knowledge of the well state. For example, we may be able to tell that a kick has occurred, but we may not be able to tell which actions have caused it, and where at the well depth the influx happened.

2.4 Existing Systems

In Chapter 1 we presented some of the state of the art systems. We did that from a wide perspectives to give the reader an overview of the research and industrial initiatives in the drilling control field. In this section we rather focus on existing approaches that are most relevant to this thesis.

The current state of affairs in the field of drilling control system seems to be dominated by two main approaches. The first approach, illustrated in Figure 2.2, strictly targets an automated control of individual devices that constitute the rig machinery. Today, almost every rig device is controlled by the driller from the *drilling control station*. Consequently, the role of the human task has been moved from a painful and dangerous job on the drill-floor to a machine steering kind of work. The main problems with today's drilling control system are directly related to the machine steering skills, the concentration and analytic abilities of the driller. The driller is required to operate different machines, follow a drilling program,

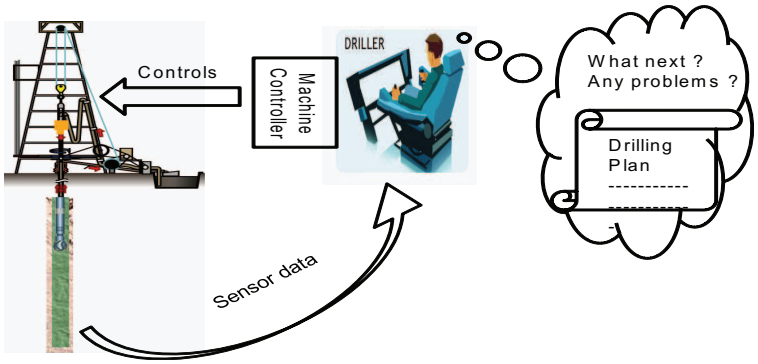


Figure 2.2: Common Drilling Control

and above all understand what is going on in the well. Note that the well state could have an influence on the machine steering. For example, when drilling a well with narrow pressure margins (pore and fracturing pressures), it is vital to move the drill-string in a smooth movement. A rapid drill-string movement causes pressure pulses that can go above fracturing or below the pore pressure, and thus causing a well incident. The second approach, called *drill-by-wire*, aims at

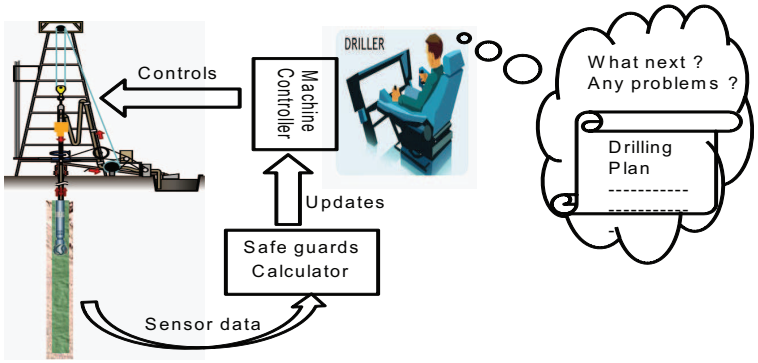


Figure 2.3: State of the art Drilling Control Setup

relieving the driller from the concern of the well, enabling the driller to keep all the focus on the rig machinery. This approach is based on sensing the environment and providing continuous feedback to the control systems, as illustrated in Figure 2.3. The feedback information is usually in the form of safe guards, such as maximum axial velocity

and maximum rotational speed.

The Drilltronics system [76, 77, 50] is an implementation of this approach. Here, the control is still left to the driller, but the action freedom is reduced when there is a risk of damaging the well or the rig. Thermo-hydraulic and mechanical models are fed with sensory data to obtain appropriate control safe guards that account for the well state. Typically, the drill-string velocity, rotational velocity, and the mud flow-rate are limited in order to avoid well incidents like formation fluid influx, hole collapse, formation fracturing, cuttings accumulations etc.

Even though the *drill-by-wire* approach solves a significant problem, drilling a well remains dangerous and costly, and does not enhance machine control itself. Machine control can still be improved in order to reduce the risk of machine incidents as mentioned in Section 2.3.

Chapter 3

Enabling Autonomous Drilling Control

In Chapter 2 we presented some existing drilling control systems. In this chapter we discuss their limitations and suggest directions to improve them.

3.1 Introduction

In Chapter 2, we presented a type of drilling systems called *drilling by wire* [76, 77, 50]. The *drilling by wire* systems rely on models that estimate the dynamics of the well and generate operational safe guards and alarms.

The architecture of *drilling by wire* systems is composed of different components that are organized in abstract layers as shown in Figure 3.1. The figure shows two sets of components: machine operability, and well surveillance.

In machine operability components we find the drilling control station which is the Human Machine Interface (HMI) used by the driller to issue control commands. These commands are sent to different device controllers via the command interface. Typically, the driller chooses a target flow-rate, which is then sent to the mud pump controller via the command interface. The mud pump controller perceives the target flow-rate as a reference point and attempts to

obtain it.

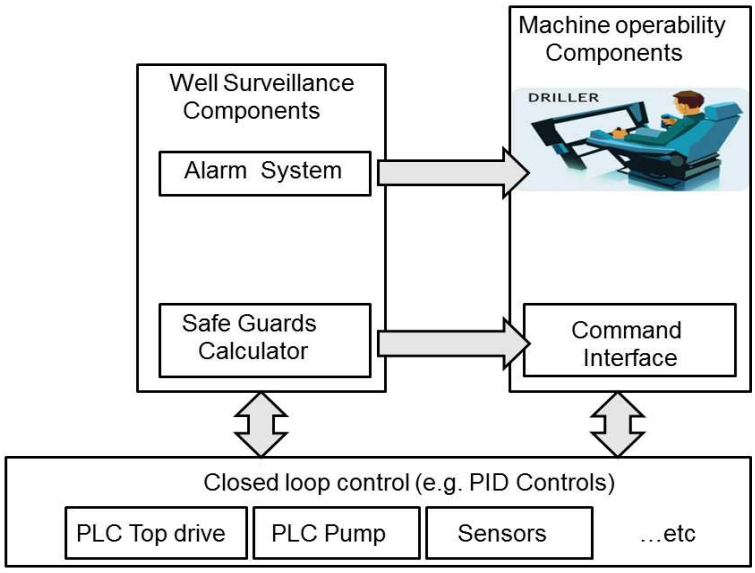


Figure 3.1: Current system architecture

In well surveillance components, we find alarm systems, and safe guard calculators. The role of alarm systems is to analyse sensor readings and generate alarms when critical situations are identified. A level below alarm systems, we find safe guards calculators, which also use sensor readings but this time to communicate the operational margins to different device controllers via the command interface. For example, the maximum flow-rate, and maximum drill-string velocity could be dynamically sent to the pump and the Draw-works controllers.

The promising results obtained so far suggest an extension of the *drill by wire* technology, to obtain safer and more autonomous drilling control systems. This can be done by identifying well-related incidents as they occur, and generate immediate or remedial actions rather than just safe guards or alarms. For example, when a gas kick is identified, the drilling fluid should be circulated down hole in order to increase the pressure and eventually contain the kick. Using generic terms, there is a need to move from an alarm system to an automatic reaction to incidents.

Unfortunately, allowing a system to execute operations on the machine control requires significant functional integration efforts. Returning to the gas kick example above, circulating the drilling fluid is only possible under some conditions, namely: the top-drive is connected to the drill-string and that the IBOP is opened. Thus, the top-drive has to be lowered to the drill-string, and attached to it, followed by a deactivation of the IBOP before the mud pump can be started.

In other words, even if we could anticipate well incidents, we cannot automatically react to those incidents. This is because to obtain the desired reaction we may need to execute a sequence of intermediate actions. However, today's drilling control systems cannot provide such action sequences. More precisely, existing drilling control systems lack control supervision.

In the supervisory control field [75, 56, 70], a control supervisor is defined as an instance capable of limiting the actions on a given system to only those permissible. In today's drilling control systems, the supervisor is the driller, and he/she is free to choose which operations to perform. That is, the driller is the one who can enforce the legal control of the system.

In addition, different incidents could require conflicting reactions, and could occur at relatively the same time. For example, a gas kick requires the circulation of drilling fluid, while a pack-off requires stopping the mud pump. To cope with such conflicts we introduce a safety process scheduler component which main task is to coordinate safety reactions.

To summaries today's drilling control systems we can say that they lack both control supervision and automated reactivity to incidents.

3.2 System Components

To cope with the limitations of existing drilling control systems, we introduce two components to the architecture: A command controller and a safety process scheduler which are shown in Figure 3.2. We place the command controller between the control station and the command interface. This component will act as a supervisor and

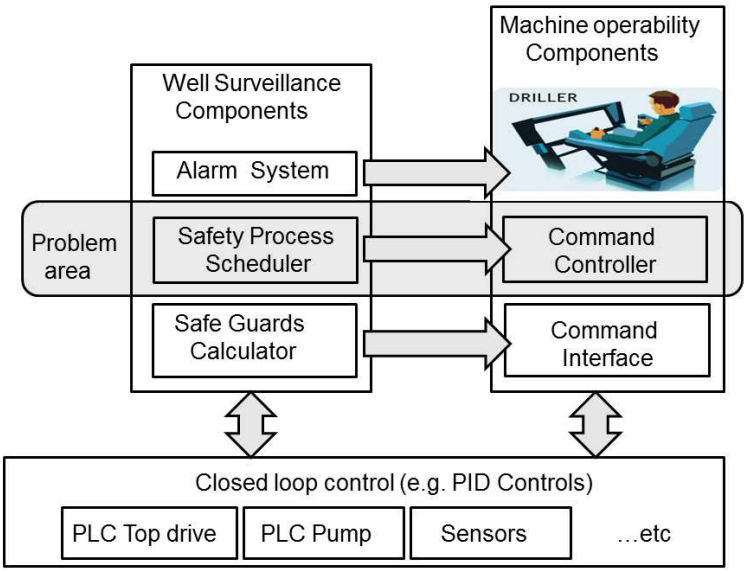


Figure 3.2: Extending the current architecture system

makes sure that the issued commands obey a certain legal behaviour. On the other hand, we place the safety process scheduler as part of the well surveillance components. This scheduler has the responsibility of regulating safety reactions when incidents occur. The command controller and the safety process scheduler are discussed in the following sections.

3.2.1 Command Controller

The main purpose of the command controller is to reduce machine related incidents. Because these incidents are mainly due to a poor controllability of the drilling equipment, the rig piloting needs to be both easier and safer.

The command controller aims to provide supervisory control to the drilling control station. That is, based on the actual state of the rig; it computes a set of permissible operations and provides them to the drilling control station.

We argued in Chapter 1 that our system can be abstracted to the DES paradigm. However, the complete drilling control system

is a combination of a continuous system part and a DES part. In this thesis, we are primarily interested in the DES part, where the command controller is its realization.

To clarify this point, the DES part of the system involves those actions that the driller takes, which in turn are of two types: on/off actions, and level actions. For example, the power-slips activation is an on/off type, while the mud pump flow-rate, or drill-string rotation are level types.

From a DES perspective, the task of the driller can be assimilated to assigning values to different control variables. The problem becomes to always ensure a legal assignment of those variables. One can thus say that the driller's task is to evolve the system from one state to another by means of actions, which is the domain of DES.

To realize the command controller component we need a DES model that captures the behaviour of the rig. It should also be possible to verify that the provided model satisfies certain properties. Questions that one would like to answer about a system are typically, whether some particular states could be reached? Whether a deadlock could occur? Is the system live? Is the system finite or infinite? etc. One of the problems we address is on how to obtain such a model.

When choosing a DES modelling formalism we have to consider two concepts: The formalism modelling power and its decision power. The modelling power describes the ability of a given formalism to capture the dynamics of systems. The more the formalism can model the higher is its modelling power.

On the other hand, the decision power describes the ability of a given formalism to determine the properties of its models. That is, the easier it is to analyse models of a given formalism the higher is its decision power.

In general, there is a common agreement in the research community, that the higher the modelling power of a given formalism the lower is its decision power, and vice versa [107].

There exists a plethora of DES modelling formalism each with distinct benefits and drawbacks with respect to modelling and decision power [107, 99]. The chosen formalism must offer sufficient modelling power for capturing the rig dynamics, and sufficient decision power for deciding model properties of interest such as deadlock, live-lock,

reversibility, liveness etc.

In the different DES formalisms we find Petri nets [108] particularly interesting. First, because they seem to fit well to our modelling needs, since they can explicitly represent the relations between actions and variable assignments. Second, because the Petri net theory is rich and is well studied in terms of decision and modelling powers. However, our choice of using Petri nets will be discussed further in Chapter 4.

Unfortunately, and to the best of our knowledge, none of the Petri nets classes today offer at the same time, sufficient modelling and decision power for our problem. We therefore propose a new class of Petri nets that answers our needs of capturing the rig dynamics in a DES model.

3.2.2 Safety Process Scheduler

The main purpose of the safety process scheduler is to reduce well incidents.

In Chapter 1, we mentioned that we seek to obtain a separation of concerns between what can be done with a rig, and what the well can support. More precisely, we seek a separation between the control of the rig and the control of the well. This claim is based on the following observations:

1. The basic functioning of the drilling rig could be captured using a DES formalism such as Petri nets.
2. The well dynamics are often hard to model using DES abstractions, as they require continuous system modelling. When the phenomena is well understood, physical models can be used. On the other hand, when the phenomena is not well understood, machine learning techniques are more common [116].
3. The multi-disciplinary aspect of the drilling industry is such that the actor realizing the machine control system is not necessarily the one predicting well incidents. This means that solutions for incident prediction and management must be integrated with the control system.

In our approach, we gradually include entities that are capable of detecting and reacting to incidents. We call such entities *reactive*

processes. A *reactive process* observes the well state for a particular incident, and has a *goal* to achieve when the incident occurs. For example, a *reactive process* could observe the well state and trigger when a kick occurs, while its *goal* could be to have the mud pump at some given flow-rate. By including another *reactive process* for handling well friction for example, or stuck pipe, we will enhance the overall capability of the drilling control system.

The role of the process scheduler component is to coordinate *reactive processes*. It provides a framework that allows the system designer to compose with such processes, and cope with eventually conflicting goals. We say that a process conflicts with another when their goals conflict. For example, if one process requests drill-string rotation, while another process requests the activation of the power-slips, a conflict occurs, because it is not possible to have both responses at the same time.

For identifying conflicting goals, we propose to associate every goal with a set of rig states. For example, the goal of having the mud pump running has many rig states that satisfy it. It could be with or without drill-string rotation, and with or without drill-string elevation etc.

Furthermore, applying basic set operations on the set of states, allows us to determine the different relations between goals. Typically, two goals conflict if they have no common states in their sets. Likewise, two goals are equivalent when they share the same states.

Finally, the process scheduler exploits the knowledge about conflicting goals to decide which *reactive process* could immediately obtain its *goal* and which should wait.

3.3 Chapter Summary

Existing drilling control systems lack supervisory control, and reaction to incidents. In order to cope with those limitations we suggest to include two components: a command controller, and a safety process scheduler.

The command controller aims at enforcing a legal control of the rig. It does that by computing a set of legal actions and provide them

to the drilling control station.

The safety process scheduler provides a systematic approach for handling well incidents. It coordinates the reactions of different *reactive processes*, where each of them is responsible for handling a specific incident.

We present the theoretical foundation of the command controller in Chapter 5 and Chapter 6, while in Chapter 7 we present the theory behind the safety process scheduler.

Part II

Theoretical Foundation

Chapter 4

Literature Review

This chapter gives an overview of the theoretical context. Based on the objective of this thesis we present the related theoretical domain, and choose a direction to follow.

4.1 Introduction

In the previous chapter we stressed out that wells could be highly unpredictable, and that capturing a precise behaviour of the well can be very difficult if not impossible. This fact leads us to the hypothesis that we can not have a complete model that captures the exact behaviour of the well, but we can have appropriate responses to critical well behaviours. One can roughly say that we don't know which actions cause which incidents, but we may know what to do in case of incidents.

As a remainder and from a drilling control perspective this thesis proposes a theoretical framework with the following objectives:

1. A modelling approach for safe machine control.
2. A modelling approach for handling incidents.
3. A modelling approach for automated drilling program execution.

Obtaining the above mentioned objectives requires a look into different theoretical domains. We shall in this chapter consider the following domains: Discrete Event Systems, Reactive Systems, Petri

nets, Process Algebra and Emergent Behaviour. These domains will be shortly presented and related to our problem.

4.2 Discrete Event Systems

We have so far presented this research from a drilling automation point of view. We shall now relate it to the Discrete Event Systems (DES) paradigm. In DES we find two basic notions: the notion of state, and the notion of event. The dynamic of a system is encoded in the relation between the states via events, and is driven by the occurrences of events [111]. Inspired by [56, 74] we explain the application domain of DESs by contrasting them to continuous systems.

Informally, continuous systems concern the modelling of continuous behaviours expressed by continuous variables. Traditionally, control theory has been concerned by providing models of the state evolution over time, usually using difference or differential equations. From an abstract point of view, given a variable $v = y_i$ at some initial stage, and a final value $v = y_f$, the problem is then to evolve v from y_i to y_f. This concept is usually exploited in closed loop control, which has also the problem of obtaining a discrete representation by sampling v into values at discrete time intervals. The continuous system paradigm is thus used to model a continuous behaviour and provide a discrete representation of it.

DES are on the other hand discrete in their nature. For example the transition from $v = y_i$ to v_f is caused by the occurrence of an event. For a DES model which contains a set of events $\Sigma = \{e_1, e_2, ..., e_n\}$, the possible sequences of events that can occur define the system language, and describe the overall dynamic of the DES in question. When the system states are expressed using a set of variables $S = \{v_1, v_2, ..., v_m\}$, the occurrences of events capture the transition between different value assignments of S.

When we regard the set of variables of S as control variables of individual parts of a system, the picture becomes clear. We could have a high level DES model acting as the logic control of the overall system, and which issues commands to the low level controller that

uses continuous control equations for each individual control variable. The high level is thus expressed in terms of events and states while the low level is expressed in some differential equations. In this thesis we are concerned by the high level part, which is usually formalised in the supervisory control theory.

4.3 Supervisory Control

Supervisory control [111] proposes a framework for designing a control components called *supervisor* that has the function of maintaining the system dynamics within the legal behaviour. For a DES, the challenge of obtaining a *control supervisor* is directly related to the possibilities of analysing its DES model.

To understand the purpose of the supervisory control theory we suggest a basic example. Consider a driver in a manual gear car, he/she knows when to use the foot brakes, when to apply the clutch to change gears, and when to use the hand brakes. In fact the driver is taking actions which are translated into set-points and further sent to the car engine. However, the driver is free to use the hand brakes while driving 120 km/h in the high way. The role of a supervisor would be to avoid such things from happening by simply disabling the hand brakes when conditions do not allow it. A supervisor is correct or not in terms of specifications. For example, one specification could be that it should not be possible to use the hand brakes when the car is in motion. Another specification would be to not use the gears if the clutch is not applied. The supervisor in this case can be seen as an encoding of the presumed correct behaviour of the system in question.

Using the car example, the driller will operate the rig as the driver drives the car. A supervisor contract in drilling control would be to enforce a legal operability of the rig by means of the enabling and/or disabling actions.

In the supervisory control theory the system in question is usually seen as a language, originally represented using automata [111, 25]. Petri nets [108] were later on exploited to express a larger family of systems [75, 56, 70]. Given that the original system has a language L_o,

a supervisory model defines a sub-language $L_s \subset L_o$ which satisfies the specified legal behaviour, that is, L_s is a sub-language of the overall language L_o. The legal behaviour is also defined using two methods, string avoidance and state avoidance [111, 70]. String avoidance expresses the legal behaviour by means of events ordering (strings). State avoidance is expressed in terms of which states should be avoided.

Supervisory control problems are often presented as synthesis problems: Given a plant model that exhibits an unsatisfactory behaviour, and a set of specifications, modify the model in such a way that the specifications are satisfied. The vision behind this idea is to automatically generate controllers that satisfy the specifications. Obviously, this means that the synthesized model is verifiable for its specifications. The problem is that not all systems can be captured in verifiable models. Some systems exhibit behaviours that require models with a high descriptive power, which unfortunately is associated with low decision power and thus hard to analyse.

We do not address the synthesis problems, but rather address the model checking problem, which is as follows: Given a model and a set of specifications, we have to answer whether the model satisfies the specifications. The required DES model that needs to be captured is the rig behaviour only. The behaviour in this case is an explicit coordination model of the different rig equipments. We stress the difference between coordinating the operations of a set of equipments and their interaction with the outside environment. The first one aims at providing a description of the different actions that can be taken, and eventually the different sequences of some high level tasks. While the second is governed by physical laws describing the behaviour of the environment.

Modelling the interaction between the different rig equipments, and also between the equipments and the well, means that we are dealing with reactive systems, or more precisely reactive DES.

4.4 Reactive Systems

DES refer to systems which state changes are controlled by events. A control supervisor aims at enforcing a certain behaviour on a DES. Reactive systems [65] refer to systems that are composed of several interacting subsystems. In the reactive system approach, the behaviour of the overall system is a result of the behaviour of its subsystems and the interaction between them.

The role of Reactive systems is not to produce or to reach any final result, but rather to maintain an ongoing interaction between one or several systems and their environment. This view of systems seems to be widely accepted as witnessed by [3, 13, 66, 63]. Reactive systems are per definition concurrent to each others, and the challenges are related to the study and analysis of such concurrent systems. If traditionally the correctness of a model was related to study of its properties, the correctness of a reactive system adds to the picture the interaction between the models.

The reactive system field of studies has been and still is an active research area. The different programming languages such as Esterel [14], Lustre [24], signal [51], statecharts [66, 67], Argos [93] witness this work. The modelling of reactive systems finds its foundation mainly in the Process Algebra theory CCS [95, 96], CSP [69], π-calculus [97], Promela [71] and more. As for which formalism one should adopt for a given system will be addressed in the next section. In this section we point out the main idea behind the reactive system approach in general, and how it fundamentally applies or not to our problem.

The underlying model of a reactive system behaviour is a transition system [85], which is a model that captures the relation between different states and the transitions that cause state changes. Provided a transition system that describes the reactive system behaviour, behavioural properties are specified using temporal logic [92, 110], and verified on the behavioural model. This system abstraction is very useful when applicable, but unfortunately its application is sometimes difficult, in particular on the expression of the desired behaviour.

To be more precise, when a system is composed of several interacting components, the reactive system abstraction seems to be

an appropriate approach. The problem however lies in the interaction with the environment. It is for instance mentioned in [65] that the interaction between a given process and its environment is similar to the interaction between a process and another process, since the environment itself can be seen as a parallel process. This claim seem a little too simplistic. The goal of a behavioural model of, say, a pressure valve, an elevator or a radiator is eventually used to provide a real implementation. On the other hand the behavioural model of the environment is in the best case a good approximation of the actual physical phenomena, but the real environment often exhibits a behaviour that the model did not account for.

The relation between drilling control and the *reactive systems* is to view the rig as one system and the well as another system, such that the overall systems is defined by both in interaction. An important issue is on the assumption behind such systems, which states that a behavioural model of each of the systems and its environment must be provided. We can unfortunately not rely on this assumption, because we know that an explicit behavioural model of the well is difficult to obtain.

The well is thus assumed highly unpredictable, and when undesired well events occur, the rig should respond by performing or inhibiting some actions. The unpredictably of the well reactions to rig operations, makes it hard to tightly couple possible sensory reading to control operations, as the analysis of the correctness of such a model becomes almost impossible by model checking techniques.

On the other hand, the rig itself is composed of several interacting components. These components combined do form a reactive system. How to capture the rig behaviour in a model that can be checked is a question that will be addressed. In fact, answering this question means that we need to choose a modelling formalism that answers our needs. We choose Petri nets, and the next section explains why.

4.5 Petri Nets

Petri nets stand for a generic name of a mathematical tool for the modelling and the analysis of DES [25, 74, 99, 87, 75]. Petri nets

were first introduced by Carl Adam Petri [108] as a graphical tool to cope with process concurrency and synchronization. They have since been an active research field with significant theoretical results and numerous applications. At first glance, Petri nets have the particularity of capturing the behaviour of systems in an easy and intuitive manner. But most importantly is the analysis techniques they offer, as they provide a good compromise between modelling and decision power [107]. These two aspects make of Petri nets an excellent tool for engineers, because a system that can be captured in a Petri net model, can as well be analysed using the Petri net theory.

The ability of Petri nets to model and formally verify properties of systems has been exploited in many application areas. These are related to the modelling and simulation of biological processes [113, 100, 27, 64, 122, 121, 68], or communication protocol verification [39, 58]. We can also find application in work-flow and business process management [133, 132, 131], or in industrial manufacturing systems [34, 139, 60]. Petri nets are also used in performance evaluation of systems [124]. Other forms of Petri nets, called continuous Petri nets have been exploited to model and analyse hybrid systems [30, 45, 6, 54]. These references are just few that illustrate the broad applications of Petri nets. There is in fact a great body of work related to theory and applications of Petri nets, with more than 8500 bibliography entries according to [59]. However more applications can be found in [59, 141].

Petri nets provide a way of expressing transition systems. They are a tool for establishing relations between transitions and places. Obtaining a Petri net model basically consists on defining places, transitions, and links between these. From a modelling perspective, they provide a much more compact model than the traditional automata. The reason is that a designer is not requested to explicitly enumerate all the states, and all transitions between states, but rather express the relation between some places and some transitions.

4.5.1 Automata and Petri Nets

Both Automata and Place/Transition nets (P/T nets) can be used to model DES. In Automata this is done by explicitly enumerating the

possible states and interconnect them with the possible transitions. The effort required to design Automata is what constitutes their main disadvantage, at least from a practical point of view. From a theoretical point of view Automata have a very rich theory behind, which is closely linked to graph theory, together they offer a number of tools for systems analysis. An objective comparison between P/T nets and Automata is proposed in [56, 25].

This comparison was based on some key criteria such as language expressiveness, modular model building and decidability. The P/T nets language is larger than Automata, meaning that Petri net can model a larger set of systems than Automata do. This is mainly due to the fact that P/T nets can express infinite systems. On model-building, P/T nets are much more natural when it comes to capturing the concurrency in DES, since this tend to become complex in Automata. Automata are better than Petri nets on problem decidability. This fact reflects the well known dilemma between decidability and descriptive power. However, P/T nets provide a sufficiently high decision power, as most problems of interest are decidable on this class of nets, but often with high complexity. It has been shown that the reachability problem for example requires at least $2^{O(\sqrt{n})}$ space [44, 107, 103], but for average size P/T models this complexity is often irrelevant.

4.5.2 Grafcets and Petri Nets

As mentioned earlier, fundamentally Petri nets are composed of places, transitions and relations between these. Depending on the system in question, one could give some concrete interpretations to these elements. Some interpretations have in fact become standard modelling tools such as grafcets [31, 32, 7] which further evolved to the Sequence Function Chart (SFC) standard [88]. Inspired from the 1-safe class of Petri nets, an interpretation was given to places to describe a particular step of the system, while transitions are used to advance the system to another step.

Even though SFC are widely used, they have a limited modelling power, compared to P/T nets. They manage to capture logical condition but can not capture counters. That is, they can model

whether a variable is set or not, but they fail to model the actual value of a given variable.

Another purely subjective difference is on the modelling technique, where in Grafcet the designer is asked to think of the system in terms of sequences of steps. This is not necessary the case in Petri nets, since a model can be designed by focusing on the transitions and define their pre-conditions and post-conditions. The sequentiality becomes an emerged property, and not an explicitly modelled one.

However, modelling convenience is often the means by which engineers choose their modelling formalisms. As we always seek more elegant ways to model and handle problems, modelling convenience is also important. Here, and to the best of our knowledge the choice remains between Petri nets and Process Algebra languages.

4.5.3 Process Algebra and Petri Nets

As mentioned earlier Process Algebra languages such as CCS [95],π-calculus [97] and CSP [69] are also very popular for modelling reactive systems. They also propose different variations, which have their strengths and weaknesses [46]. Fundamentally, CCS and CSP are the ancestors of Process Algebra, while π-calculus inherits mostly CCS and improves it [10].

What is important to retain in modelling a system using these approaches is that a system is expressed by the actions it takes. The questions that such a model answers are based on what sequences of actions it can generate. In such case, Process Algebra languages are quite elegant in modelling systems. On top of that, Process Algebra are explicitly built on the notion that processes can be composed using other processes, which provides a structured way of building systems out of sub systems. The whole idea behind Process Algebra is to combine and compose with processes that may also communicate with each others, and verify that these systems satisfy some properties.

Petri nets can also view a system by means of the sequences of strings it generates [62, 78]. Regarding a system from the language it generates is to a large extent exploited in supervisory control with Petri nets [56, 75] but they somehow fail in capturing the modularity of systems. For example, a system A which is composed of B and C,

will result in a Petri net model tightly linking both, which is much more decoupled in a Process Algebra model. In addition, Process Algebra adopt the notion of communication between process in a formal way, as it is clear upon which actions taken from a given process that are communicated to the other [46].

There are however differences among the Process Algebras on their view of communication. CSP for example operates on a broadcast communication, such that when a process takes an action it is visible to all the others. On CCS however, it is adopted a point to point communication, such that an action taken by one process is visible only to processes needing it.

Even though these modelling languages are different in their semantics, they share the same view of systems. In these languages a system is composed of several communicating processes, and processes are composed using actions (atomic actions). The underlying model of these Process Algebra are transition systems, where some are limited to finite transition systems such as Promela [71], while others like CCS, CSP and π-calculus can capture infinite transition systems. The specifications are represented with temporal logic formula [110, 92].

Petri nets can also capture the communication between processes, it is just more cumbersome. On the other hand Petri nets capture another type of information, which is the explicitness of the states. In Process Algebra a state contains information about which actions lead to it, and what actions are possible from it. But in Petri nets and in additions to that, it can also tell what a state consists of. If a state is for example composed of set points variables, knowing the values of these variables at a given state is sometimes important. In fact this difference is our main justification of using Petri nets rather than Process Algebra, because for us what ultimately counts is that the right set points are assigned to the right variables.

Putting side by side Petri nets and Process Algebra is a large topic that has partially been addressed in [37, 15, 57], and is beyond the scope of this thesis.

4.5.4 Petri net Classes

After having justified the choice of using Petri nets, what remains is to choose a class among the many Petri nets classes that fit best to our needs for modelling the rig dynamics. The fact that we may need to assign set-points such as flow-rates, or rotations with high precision means that we are dealing with large granularity, which leads us to the assumption of infinite systems.

A natural candidate is thus the P/T nets class. This class is attractive because it provides a good compromise between modelling and decision power. P/T nets is probably the most studied, and can be seen as the border line of what can be modelled by Petri nets and still maintain high decision power. The complexity of some systems often requires large models, and it appears that this class is not always convenient for designers, because large models tend to become intractable. This issue calls for higher level models, that not only capture a large family of systems but also keep their models compact.

In high level Petri nets the most general class is Coloured Petri nets (CPN) [138, 81, 80, 79]. What this class really offer, is no more nor less than a better modelling convenience. Theoretically, CPN do not increase the modelling power nor do they increase the decision power of P/T nets [106]. In fact CPN are to P/T nets what high level programming languages are to assembler. This means that complex systems for P/T nets can be elegantly captured in CPN. It also means that no real computational capabilities are gained using CPN.

This result roughly means that if a simple model can not be captured with P/T nets, it can not be captured with CPN neither. What is missing is a true extension that increases the modelling power. That is, adding some basic capabilities so that one can design a system which otherwise could not be designed. The capability we are looking for is the ability to test for zero. The PTI nets class [4] (PTI) does capture our modelling needs, but this class has been proven Turing equivalent [61, 44]. The Turing equivalence of PTI nets makes it difficult to claim the correctness of the model, since we cannot determine the properties of that model.

The Turing equivalence of PTI nets also means that algorithms

which apply to P/T nets do not necessarily apply to PTI nets. In particular the boundedness and the reachability problems, become undecidable for PTI nets.

This dilemma gave us two choices, either we look into some other formalisms, review the system from an abstraction so that we could apply something else such as Process Algebra, or dig further into Petri nets to find solutions. We have chosen to dig further into Petri nets.

Informally, we would like to express the following (v1 can be set only if v2 is zero, and v2 can be set only if v1 is zero) where v1 and v2 may be any large number.

Figure 4.1 illustrates this situation using a PTI net. This model can be read as follows: t1 can fire as many times as desired to fill as many tokens in v1, as long as v2 is empty. The transition t3 can also fire as many times to fill v2 as long as v1 is empty. Despite the simplicity of this model, there is no general algorithms that can determine its properties (Boundedness, liveness, marking reachability etc.).

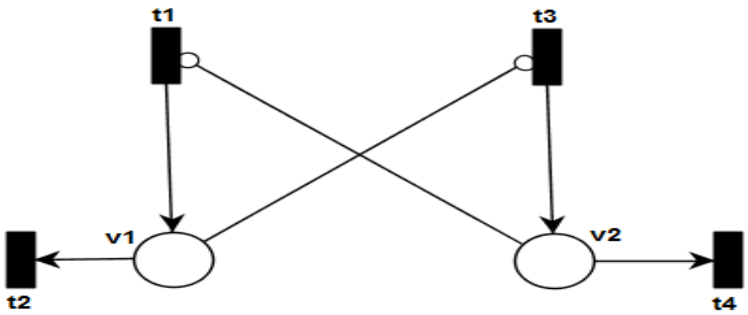


Figure 4.1: Basic P/T net with inhibitors

The Turing equivalence of PTI nets is clearly very important, but also frustrating. Because in reality, and as Figure 4.1 shows, it is sometimes necessary to use inhibitory arcs, even for capturing a simple behaviour. A strategy here could be to use inhibitory arcs in some restrictive ways in order to obtain a good decision power. This strategy was already used in [23].

An important conclusion from [23] is that, some restrictive use of

inhibitor arcs could maintain a high decision power. However, the example shown in Figure 4.1 does not belong to the primitive PTI class.

Our motivation in this thesis in terms of Petri nets is thus to capture the dynamic behaviour of the rig, in a PTI net on which it is possible to decide the reachability and the path problems. For that, we define a class called Cohesive PTI nets (CPTI), on top of which we define an even more restrictive class called Mutually Inhibited Cohesive PTI nets (MICPTI). These classes are such that they contain two basic structures, where a structure is a composition of sub nets using places and transitions.

Those basic structures are inspired from real word control components for describing on/off type of variables, and level variables, as shown in Figure 4.2. The inhibitory arcs can only be applied cross these structures. Figure 4.2 presents a simple model of a motor which can be controlled using three variables: ON, OFF and Speed. Those variables are modelled using Petri net places (circles). The model captures the following behaviour: The transition Turn_OFF will set the place OFF only if the motor speed (place Speed) is set to zero. Likewise, increasing the speed is possible only when the motor is not OFF (Place OFF set to zero). Surprisingly, the MICPTI class of nets enjoys very high decision power. We show how to generate the coverablity graph of these classes, and how it allows us to decide the reachability, boundedness and the path problems. We also show how MICPTI nets elegantly captured the required rig dynamics.

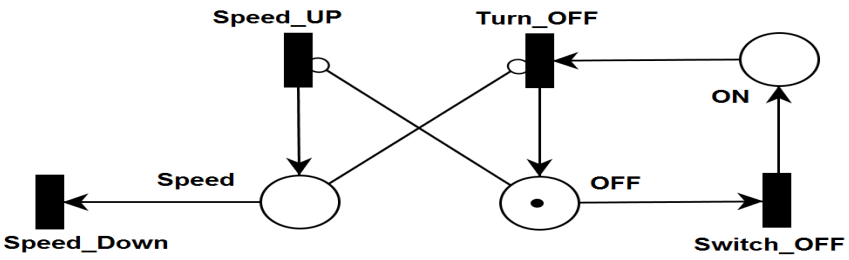


Figure 4.2: An example MICPTI net

4.6 Emergent Behaviour

We already mentioned that we advocate a separation between the rig as a system and the well as the environment. The rig system will be modelled using a specific class of Petri nets which makes it possible to analyze the system behaviour. The well dynamics will be sensed, but in contrast to the rig, its state space can not be fully determined. We suggest an approach that gradually evolves towards a controlled behaviour of the well.

Emergent behaviour in robotics refers to the idea of obtaining complex behaviours out of basic reflexes. This concept was studied by Rodney Brooks in the so-called Subsumption architecture [19]. This approach is seen as an application to Behaviourism which was introduced as a model of animal psychology [136].

In this theory, the activity of living beings is determined by a set of elementary behaviours each of which is characterized by reactions to stimuli from the environment, the overall behaviour is then composed of the elementary ones. Applying behaviourism to robots leads to a layered control based on the so-called subsumption architecture.

Typically, a robot could be built with the competence of *never hit an object*, adding a new competence *move around* should include *never hit an object*. The emerging behaviour will then be *move around and never hit an object*.

Even though a number of robots have been successfully developed using this approach, it sometimes fails in handling unexpected behaviours [12, 20]. In particular when competing modules are involved.

In our approach, the gradually added competences can be regarded as separated processes that focus on very specific aspects called *reactive processes*. For example, a robot could have a reactive process that is focused on avoiding collisions. If an object is too close, the process orders the robot to stop. Another process would compute a trajectory and order the robot to deviate its trajectory. By combining these processes the behaviours *move around and never hit an object* may be obtained.

The value that we would like to add is the ability to say something about the added competences, whether they conflict or subsume the

existing ones.

Our approach is in line with the work proposed by [20, 21] about Behavioural Oriented Design. In her work, she stressed out that designing intelligent agents is done by establishing ***WHAT*** *to do* ***WHEN****, and* ***HOW*** *to do it.* In our approach reactive processes are defined by **WHAT** to do **WHEN**, where the **WHAT** defines the *goal* of a *reactive process*, the **WHEN** defines its triggering condition. The **HOW** to do it, is on the other hand determined by the system on which a *reactive process* reacts.

For example, a process that requires the starting of the mud pump may be conditioned by a suspected too low well pressure. As for how to start the pump is limited by the state of the rig at the triggering moment, which in turn dictates sequence of actions that needs to taken for having the mud pump running.

Finally, the overall system could handle more incidents by adding *reactive processes*, and ultimately reach a satisfactory autonomous behaviour.

Chapter 5

Petri Nets

Petri nets include various classes, each having particular features. This chapter gives a short introduction to Place/Transition nets. We introduce concepts and definitions that are relevant for this work in a fairly intuitive manner.

5.1 Introduction to Petri Nets

Petri net is a modelling notion with a strong mathematical underpinning targeted at analysing discrete event systems [25, 74, 99, 87]. Petri nets were first introduced by Carl Adam Petri [108] as a graphical tool to cope with process concurrency and synchronization. They have since been an active research field with significant theoretical results and many applications.

A P/T net is a directed bipartite graph with two types of nodes: *places* represented by circles and *transitions* represented by rectangles. A place can only be connected to a transition and a transition can only be connected to a place. The connection between nodes is done using directed weighted arcs. A place may contain tokens which are represented using a black dot. The tokens can flow between different places by firing transitions. A transition can fire if the number of tokens contained in all the places that have arcs directed to it, is larger or equal to the weights of their corresponding arcs. These places describe the pre-conditions for a transition to

fire. When a transition fires, it deposits as many tokens in its output places as indicated by the output arc weights. The distribution of tokens over the places is called a marking, or state.

Figure 5.1 shows a P/T net before firing the transition t_2, and after its firing. As the figure shows, the arc between p_2 and t_1 has a weight of 4. Since p_2 has only 3 tokens t_1 can not fire. The transition t_2 can fire because both p_1 and p_2 contain enough tokens. When t_2 fires, only one token is deposited in p_3. Note that an arc with no number marked on it has a weight of 1. In this thesis we do not

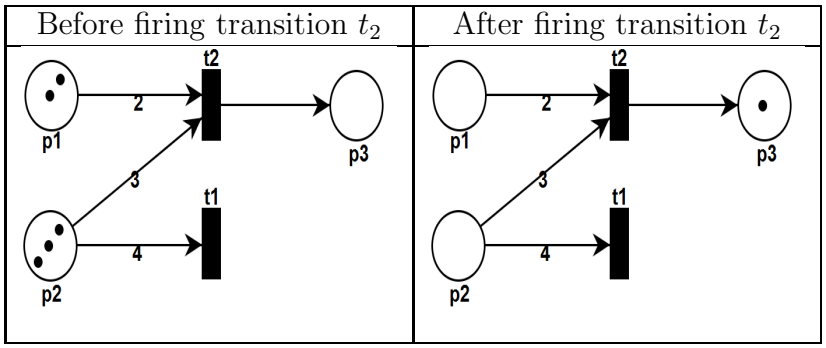


Figure 5.1: An Example P/T net with weighted arcs

consider arc weights. The term P/T nets will refer to ordinary place transition nets, i.e. P/T nets with arc weight equal to one.

Important questions can be answered about a P/T net [99, 61] such as the existence or not of a deadlock situation. If deadlocks exist, under which states could they occur? And how to get to those states? Is the system bounded? That is, is it possible for some variables to have arbitrarily large values? Is the system live? Is it possible to reach a state in which a particular transition can fire? Is every transition eventually firable? etc.

P/T nets seem to constitute a theoretical border line between what can be, and what can not be analysed[107]. This is mainly justified by that most of the questions about the behaviour of P/T nets are decidable [44], but are of high complexity. As a rule of thumb, solving P/T nets related problems usually requires $2^{(O\sqrt{n})}$. This roughly means that for relatively small size P/T nets most of the problems of interest are tractable.

The possibility to analyse P/T nets makes them to an ideal tool for modelling DES in general [25, 74, 99, 87]. However, a system that can not be modelled using P/T nets is little likely to be analysed by means of model checking techniques. One could consider simulation techniques in order to derive probable properties of a system, but this is beyond the scope of our work.

The border line between decision and modelling power has fascinated and still fascinates a number of researchers. Several extensions have been proposed to enrich the modelling power of Petri nets. Among these extensions we find Coloured Petri nets [138, 81, 79, 1](CPN). In CPN tokens are associated to colours, this makes it possible to give to the tokens different interpretations. Hierarchical object oriented Petri nets [72, 98] attempt to take advantage of the object oriented programming paradigm. Timed Petri nets [112] introduce the notion of time to model durations and delays, and are usually used for performance evaluation of systems [124, 140]. A Petri net using Coloured tokens, hierarchies, and timed transitions is called High-Level Petri net [2, 134].

Continuous and hybrid Petri nets are also extensions made to ordinary P/T nets in order to model continuous and hybrid systems [30, 45, 6, 54]. Another interesting form is found in synchronous Petri net [30], a form that attempts to include sensory data into transition firing rules.

In this thesis, we are particularly interested in ordinary P/T nets extended with inhibitor arcs [4]. An inhibitor arc is a type of arc that inhibits a transition from firing when the inhibiting place contains at least one token.

The problem of extending P/T nets with inhibitor arcs will be studied in the next Chapter. This chapter presents basic notions and definitions; it can be viewed as an introduction to Chapter 6. Section 5.2 provides an informal and a formal presentation of P/T nets. Section 5.3 presents the analysis of Petri nets properties by means of either the reachability or coverability graphs. Section 5.4 provides a short introduction to P/T nets extended with inhibitor arcs, and finally Section 5.5 summarises this chapter.

5.2 Basic Definitions

This section provides definitions and illustrations of necessary concepts regarding Petri nets. The provided definitions are inspired mainly from [99, 107]. Different concepts will be introduced gradually, supported with appropriate examples, which hopefully will make it easier for the reader to understand.

Before proceeding to definitions, we would like to clarify a few points that are related to the interpretations of tokens, places and transitions. For a Petri net model capturing a given dynamics, it is up the modeller to define what the places, transitions and tokens represent, and how they relate to the designed system.

At the risk of repetition, in this thesis Petri nets places represent variables that need to be assigned, tokens represent the values assigned to those variables, while transitions represent the actions taken for assigning the variables. We suggest a simple way to relate these interpretations to real systems by viewing the transitions as buttons on a given control panel. When a button is pushed a variables will be assigned some values. These variables could be some control variables, or some internally used ones, such as counters, or flags.

5.2.1 P/T Nets

We start by defining the P/T net; its elements and notations in Definition 5.1, followed by an illustrative example (see Example 5.1).

Definition 5.1 (Formal definition of P/T nets)**.**
P/T net is quadruple $PT = (P, T, F, m_0)$ *where:*

- $P = \{p_1, p_2, ..., p_m\}$ *is a finite set of places.*
- $T = \{t_1, t_2, ..., t_n\}$ *is a finite set of transitions.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation).*
- $m_0 : P \to \{0, 1, 2, 3, ...\}$ *is the initial marking.*
- $P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$.

Notation 5.1 (P/T nets notations)**.**

- ${}^\bullet t = input(t)$ *denotes the set of pre-places of transition* t.
- $t^\bullet = output(t)$ *denotes the set of post-places of transition* t.

- ${}^\bullet p$ *denotes the set of pre-transitions of place* p.
- $p^\bullet$ *denotes the set of post-transitions of place* p.
- $m(p)$ *denotes the number of tokens in a place* p *for a marking* m.

5.2.2 P/T net Example

This example illustrates the elements of Definition 5.1, using a P/T net model of an espresso machine. Because ground coffee deteriorates faster than coffee beans, the espresso machine allows a beans grounding of a maximum amount that is sufficient for two espressos, before an espresso is made.

Example 5.1. *Figure 5.2 shows a P/T net model of the espresso machine. The transition* t_1 *stands for the grounding of coffee for one espresso. The amount of grounded coffee is represented in the place* p_2*, where the number of tokens indicates the number of espressos that can be served. The transition* t_2 *fills the cup with one espresso at the time by placing a token in the place* p_3*, and removing one token from the place* p_2*. Making one espresso gives the possibility to ground for another one. This is represented by the transition* t_3 *which consumes one token from* p_3 *and deposits one token in* p_1*. Using Definition 5.1 we can extract the information in Table 5.1.*

Table 5.1: A P/T net definition example.

- $P = \{p_1, p_2, p_3\}$.
- $T = \{t_1, t_2, t_3\}$.
- $F\{(p_1, t_1), (t_1, p_2), (p_2, t_2), (t_2, p_3), (p_3, t_3), (t_3, p_1)\}$.
- ${}^\bullet t_1 = \{p_1\}$, ${}^\bullet t_2 = \{p_2\}$, ${}^\bullet t_3 = \{p_3\}$.
- ${t_1}^\bullet = \{p_2\}$, ${t_2}^\bullet = \{p_3\}$, ${t_3}^\bullet = \{p_1\}$.
- ${}^\bullet p_1 = \{t_3\}$, ${}^\bullet p_2 = \{t_1\}$, ${}^\bullet p_3 = \{t_2\}$.
- ${p_1}^\bullet = \{t_1\}$,${p_2}^\bullet = \{t_2\}$, ${p_3}^\bullet = \{t_3\}$.
- $m_0 = [2\,0\,0]$, where $m_0(p_1) = 2$, and $m_0(p_2) = m_0(p_3) = 0$.

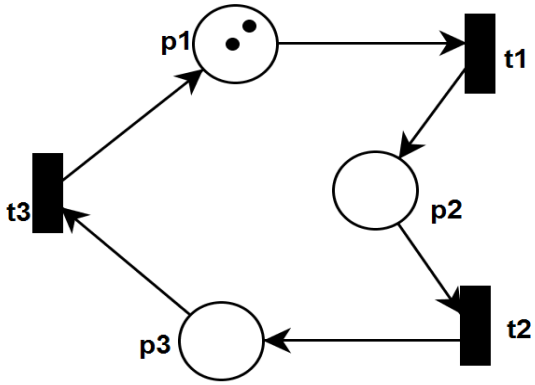


Figure 5.2: P/T net model of an espresso machine

5.2.3 Transition Firing and Sequences

A P/T net is defined by its places, transitions, relations between these, and an initial marking. The dynamic of the model is reflected in the change of its markings, which in turn is caused by the firing of transitions. In a P/T net (ordinary P/T net), a transition can fire if all its input places contain at least one token. When a transition fires, it removes one token from each of its input places, and adds one token to each of its output places. The notion is formalised in Definition 5.2.

Definition 5.2 (Transition Firing)**.**
A transition t can fire in m iff: $\forall p \in^\bullet t, m(p) > 0$. The set of transitions that can fire in m is denoted enabled(m).

When t fires: $\forall p \in t^\bullet, m(p) = m(p) + 1$, and $\forall p' \in^\bullet t, m(p') = m(p') - 1$.

We write $fire(m, t) = m'$ and read, firing t in a marking m generates m'. We write $m_a[\sigma\rangle m_b$ to denote a sequence of transition from m_a to m_b. We write $m_a[\Sigma\rangle m_b$ the set of transition sequences from m_a to m_b. We write $m_a[\sigma\rangle$ or $L(m_a)$ for the possible sequences from m_a. Consequently, $L(m_0)$ or $m_0[\sigma\rangle$ denote the net's possible sequences of a net. The set $R(m)$ denotes the reachable markings from m. The set $R(m_0)$ denotes the reachable markings of a net.

5.2.4 Transition Firing Example

This example illustrates the dynamics of the coffee machine (see Figure 5.2 and Example 5.1) by means of transition firing rules from Definition 5.2.

Example 5.2. *Figure 5.3 shows the six states that the coffee machine can be in. For example at m_0(top left figure), the only enabled transition is t_1, and firing t_1 from m_0 leads to m_1. From the marking m_1, it is possible to fire either t_1 or t_2 to reach m_2 or m_3 respectively. From m_0, the sequence $\sigma_a = \langle t_1, t_1, t_2, t_2 \rangle = \langle t_1^2, t_2^2 \rangle$ leads to m_5, and so does $\sigma_b = \langle t_1, t_2, t_1, t_2 \rangle$, both sequences belong to the set $m_0[\Sigma\rangle m_5$. Finally, the set of reachable markings is $R(m) = \{m_0, m_1, m_2, m_3, m_4, m_5\}$. The sequences σ_a and σ_b illustrate two ways of obtaining a double espresso (assuming that a double espresso stands for two single ones).*

5.3 P/T Nets Problems and Analysis

Petri nets analysis consists of methods for determining properties of the modelled system. From the field of research we can identify the following techniques for Petri net analysis: algebraic, reduction rules, and state space exploration techniques. The two first ones can be computationally very efficient, but are usually applicable to special subclasses of P/T nets only [99].

The state space exploration technique can be divided into two categories: Reachability, and Coverability analysis. Both methods suffer from the state explosion problem [130], and are therefore limited to relatively small models. Reachability analysis applies to finite systems only (bounded nets), since it consists of enumerating all the reachable markings, and the relation between these. On the other hand, the coverablity analysis, applies to infinite systems (unbounded nets), and consists of capturing an approximation of the state space in a compact graph, called the coverability graph.

The algebraic, and structural techniques are beyond the scope of this thesis. Recall that this chapter's purpose is to introduce the necessary concepts that will be used later in the thesis. We therefore

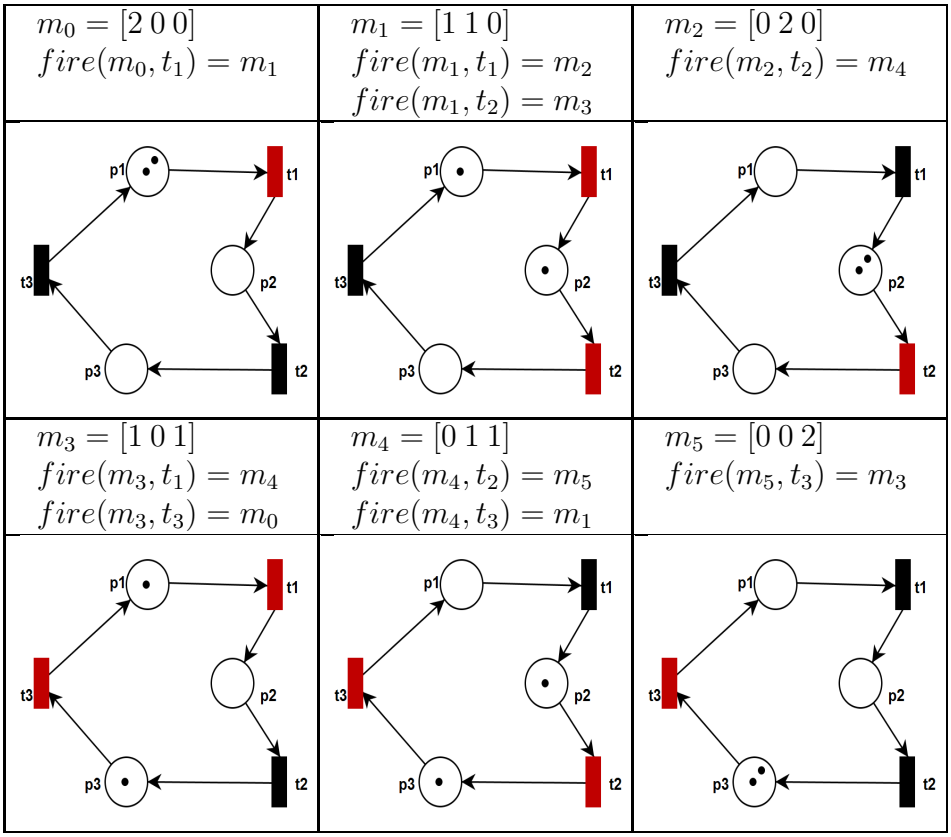


Figure 5.3: Firing a sequence of transitions

focus on state space exploration, as it is the technique used to analyse the class of nets that we shall present in the next chapter.

This section presents different Petri nets problems and shows how they can or cannot be solved on the P/T nets, using either reachability or coverablity analysis.

5.3.1 Reachability Graph

The reachability graph [99, 107, 128] is a graph where nodes represent markings and edges represent transition. Starting from the initial marking m_0, the reachability graph is obtained by triggering all the possible transition firings as defined in Definition 5.2, and keeping

track of the generated markings. Algorithm 1 in Section 5.6 takes as input a Petri net, and generates its corresponding reachability graph.

Figure 5.4 shows the reachability graph of the previously presented P/T net of the espresso machine (see Figure 5.2). Furthermore, the reachability graph in Figure 5.4 contains all the information provided by Figure 5.3, because from a given marking it is possible to determine which transitions could fire next, and which marking could be reached. However, when the set of reachable markings is infinite, Algorithm 1

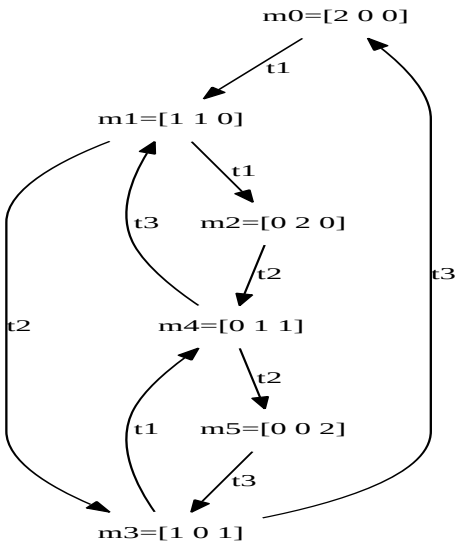


Figure 5.4: Reachability graph corresponding to the P/T net of Figure 5.2

will not terminate, and the generated reachablity graph will grow infinitely large. In other words, the reachability graph is finite only for bounded nets. A place p is bounded if it does not contain more than k tokens in any reachable marking, likewise a net is bounded if all its places are bounded. A formal definition is given in Definition 5.3.

Definition 5.3 (Boundedness)**.**
A place $p \in P$ is bounded iff: $\forall m \in R(m_0)$, $\exists k \in \mathbb{N}$ such that $m(p) \leq k$. A P/T net is bounded iff: $\forall p \in P$, and $\forall m \in R(m_0)$, $\exists k \in \mathbb{N}$ such that $m(p) \leq k$.

5.3.2 Coverability Graph

The reachability graph can in theory be generated for any Petri net, under the condition that the net is bounded. On the other hand, the coverability graph applies to unbounded nets as well, but does not seem to apply beyond the class of P/T nets.

Informally, an unbounded net is a net which has at least one place that could contain an infinitely large number of tokens. This large number is represented using the symbol ω. Furthermore, an extended marking is a marking containing at least one ω symbol, and can be viewed as a compact representation of an infinite set of markings. A formal definition of extended markings is given in Definition 5.4.

Definition 5.4 (Extended Marking)**.**
For any constant $a \in \mathbb{N}$, $\omega + a = \omega - a = \omega$, $a < \omega$ and $\omega \leq \omega$.

The marking m is an extended marking iff: $\exists p \in P$ such that $m(p) = \omega$.

Definition 5.5 (Covering)**.**
A marking m' covers m ($m \leq m'$) iff: $\forall p \in P, m(p) \leq m'(p)$.

A marking m' strictly covers m ($m < m'$) iff: $\forall p \in P, m(p) \leq m'(p)$ and $m \neq m'$.

Definition 5.6 (Coverability Graph)**.**
Let G=(V,E) be a graph, where V is the set of markings and extended markings, and E the set of edges connecting the markings.

For a net N, G constitutes a coverability graph iff:

$\forall m \in R(m_0)$, $\exists m' \in V$ such that $m \leq m'$. This means that every reachable marking m is either explicitly present in G or is covered by some marking in G.

The algorithm used for generating the coverability graph originates from [84] (Karp and Miller procedure). An attempt to optimise the original version was proposed in [48], and later on proven incomplete in [53, 49]. The coverability algorithm is also of particular interest in this thesis, as it will be used as a basis for analysing the Petri net class that will be presented in the next chapter.

The coverability graph is a finite representation of a possibly infinite set of markings. The construction of the coverability graph

is done in a similar manner as with the reachability graph, that is, by enumerating every reachable marking. Running the coverability graph algorithm(see Algorithm 2 in Section 5.6) on a bounded P/T net generates the reachability graph, while running it on an unbounded P/T net generates the coverability graph. The main drawbacks with the coverability graph algorithm is that it is limited to P/T nets only and that it uses a so-called acceleration function which makes it run slower.

The acceleration function is responsible for inserting the special symbol ω when there is evidence that a place can contain an arbitrarily large number of tokens. Algorithm 2 works as follows: Starting from the initial marking, the algorithm determines the next marking to explore. Each marking passes through the acceleration function for further processing. So, if the current marking m' is such that there exists a marking m'' on the path from the initial marking m_0 to m' and that $m'' < m'$ then a ω can be inserted in each place p where $m''(p) < m'(p)$.

5.3.3 Coverability Graph Example

This example shows how Algorithm 2 generates a reachability and a coverability graph for a bounded and unbounded net respectively.

Example 5.3. *For the sake of space, we use a slightly different P/T net than the one proposed in Example 5.1. We consider that the espresso machine can only ground for one espresso, rather than two before an espresso is made (see Example 5.3). Figure 5.5 shows two P/T nets of the espresso machine. The net at the left side is a bounded net, and is very similar to the previous model in Figure 5.2. The model at the right side includes an additional place p_4, which acts as a counter. Each time an espresso is made by firing the transition t_2, the number of tokens in p_4 is incremented by one. The P/T net at the top right side is thus unbounded. Algorithm 2 generates a reachability graph when run on the top left net, and coverability graph when run on the top right net.*

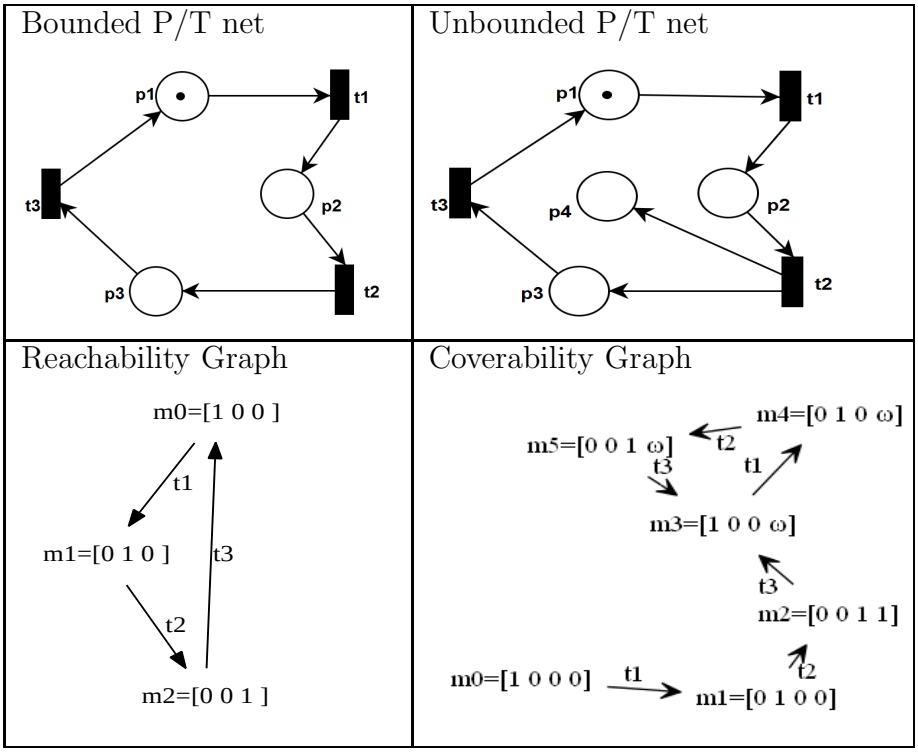


Figure 5.5: The coverability graph algorithm applied to bounded and unbounded P/T nets

5.3.4 Boundedness Detection

Determining the boundedness of a given system or of some variables in the system is sometimes desired. When designing a system, some variables are expected be bounded, while others are not. For example, in the espresso machine from Example 5.3, the place p_4 (counter) is meant be unbounded, while the place p_2 is meant to be bounded.

When a system is modelled by a P/T net, the boundedness property is decidable. Decidability is obtained by generating the net's coverability or reachability graph using Algorithm 2, and by inspecting the graph's nodes for the presence of ω. The presence of ω in a marking indicates that its corresponding place is unbounded. The absence of ω in any node indicates that the net is bounded, and the graph is actually a reachability graph [99].

5.3.5 Deadlock Detection

Firing transitions on a Petri net model evolves markings, some of these markings could be deadlock markings. Deadlock markings are usually undesired when modelling a system, but this needs not to be always the case. In fact a model could be meant to describe a non terminating process, while another could be meant to describe a terminating one. Regardless of its use, determining deadlock markings is decidable for P/T nets.

Informally, a deadlock describes a situation where no transitions can fire any longer. Furthermore, a Petri net is deadlock free, if none of its markings is deadlock. A formal definition is given in Definition 5.7.

Definition 5.7 (Deadlock)**.**
Let m_0 be an initial marking of a Petri net N and let $R(m_0)$ the set of reachable markings of N.

A marking $m_i \in R(m_0)$ is a deadlock marking iff: $enabled(m_i) = \emptyset$
N is deadlock free iff: $\forall m \in R(m_0)$, $enabled(m) \neq \emptyset$

Deadlocks can be detected by inspecting the reachability or the coverability graph for markings (or extended markings) without any outgoing edges. For example, all the graphs from Figure 5.5, and Figure 5.4 correspond to deadlock free nets, because every reachable marking has at least one outgoing edge.

Figure 5.6 shows a bounded P/T net and its corresponding reachability graph. The markings m_2 and m_3 are both deadlock markings, since they have no outgoing edges.

Figure 5.7 shows an unbounded P/T net and its corresponding coverability graph. The marking m_2 is deadlock, and so is the extended marking m_3. In this case, m_3 represents infinitely many deadlock markings. The reason of that is that the transition t_1 in Figure 5.7 can fire as often as needed until t_2 fires which blocks to whole dynamics, and no transition can fire anymore.

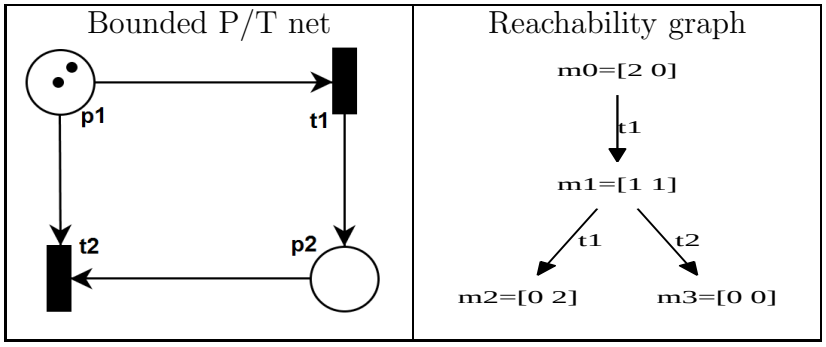


Figure 5.6: A bounded P/T net with deadlock markings: m_2 and m_3

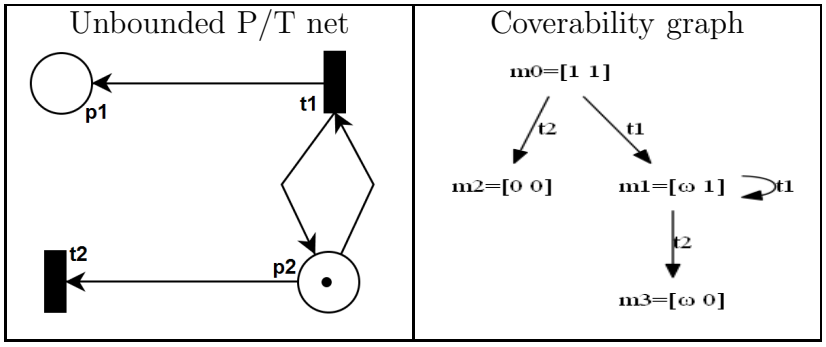


Figure 5.7: An Unbounded P/T net with deadlock marking m_2 and deadlock extended marking m_3

5.3.6 Marking and Sub-Marking Reachability

The marking reachability problem consists of determining whether a given marking is reachable or not. The decidability of this problem means that one can check whether some desirable or undesirable states can be reached.
If a marking is specified by the token values of each place from the set of places P, a sub-marking is only concerned by a subset $P' \subset P$. The decidability of the sub-marking reachability problem means that one can check whether some given state variables will ever be assigned some given values.

Definition 5.8 gives a formal definition of the marking and sub-marking reachability problems.

Definition 5.8 (Reachability Problem)**.**
Given a Petri net N with m_0 as its initial marking and $R(m_0)$ as the set of its reachable markings. Let P be the set of place of N, and $P' \subset P$.

- *The reachability problem of a marking m is: Does $m \in R(m_0)$?*
- *The sub-marking reachability problem of m' specified over P' is: Does there exist $m \in R(m_0)$, such that $m(p_i) = m'(p_i)$, $\forall p_i \in P'$?*

For a bounded Petri net, the marking and the sub-marking reachability problems can be solved by inspecting the set of reachable markings. A marking m is reachable if it belongs to the reachability graph. For example, using the reachability graph of Figure 5.4, one can ask whether it is possible to reach a marking $m = [1\ 1\ 1]$? The answer is no, since no marking in the graph equals m for each place.

Obviously, the sub-marking reachability problem can also be solved in a similar manner. More precisely, deciding the sub-marking reachability is a special case of the marking reachability, because when inspecting the reachability graph, we need to compare markings for a subset only, rather than the complete set of places.

The sub-marking reachability problem solves a particularly interesting situation. Consider the coffee machine model from Figure 5.2 and its reachability graph (see Figure 5.4). One could ask whether it is possible to reach a marking where a single espresso is served, which could be formulated using $m(p_3) = 1$ with $m_3 = [1\ 0\ 1]$ and $m_4 = [0\ 1\ 1]$ as answers.

For unbounded P/T nets the coverability graph does not provide sufficient means for deciding the marking reachability. For a marking m to be reachable, it has to be covered by a marking in the coverability graph. This condition is necessary, but not sufficient for m to be reachable. In other words, all reachable markings are covered by some markings in the coverablity graph, but not all covered markings are reachable.

The reason of the non sufficiency of the coverability graph for deciding the marking reachability problem lies in the ω approximation which hides some artefacts of not only on the actual reachable markings, but also on the effect of transition firings. When for

example a given place of a net can only contain an odd number of tokens, this information is hidden by ω. Likewise when a transition consumes or deposits a token in some places, this information is also lost by the relation $\omega + a = \omega - a = \omega$ from Definition 5.4.

For example, Figure 5.8 shows an unbounded P/T net and its corresponding coverability graph. Now, consider that we want to determine whether a marking $m = [5\ 7]$ is reachable. By inspecting the coverablity graph we can find that $m \leq m_1$ ($\leq$ from Definition5.4). Even though m is covered by m_1 it is not a reachable marking, the P/T net from the figure is such that p_1 and p_2 always contain the same number of tokens, which is not the case of m. Nevertheless, the

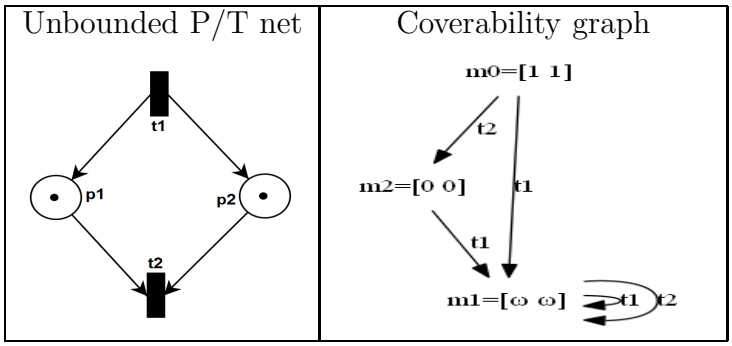


Figure 5.8: Coverability is a necessary but not sufficient condition for marking reachability

reachability problem has been proven decidable[120, 94, 4]. So, even if a Petri net can have unbounded places, i.e. an infinite number of reachable markings, the reachability problem remains decidable.

5.3.7 Path

We have already mentioned that the main purpose of Petri nets is to capture the dynamic of the studied system. Since the dynamic of a system consists of transitions between states, a legitimate question to ask is whether it is possible to find a sequence of transitions from one state to another?

When the system dynamics is captured in graphs such as the reachability or coverability graph, a sequence of transition can be

referred as a path between two nodes on a graph.

In the case of bounded nets, we can generate a reachability graph. Obviously, finding a path from a source to a destination marking can be found using breadth-first or depth-first search on the reachability graph.

In the case of unbounded P/T nets, we cannot generate a reachability graph, but rather use the coverability graph. However, the coverability graph does contain enough information do determine the path between two markings. This is mainly due to ω which could hide some information about the transition between markings.

5.3.8 Home Marking and Reversibility

For a model representing a given system, it is sometimes desirable to check whether a given state can always be reached for any other state. In Petri nets such a state is called a home marking. When the initial marking of a Petri net is a home marking, then the net becomes a reversible net. A formal definition is given in Definition 5.9.

Definition 5.9 (Reversibility and Home Marking)**.**
Let N be a Petri net and m_0 its initial marking:

- *A marking m is a home marking iff:* $\forall m' \in R(m_0)$, $m \in R(m')$.
- *N is reversible iff:* $\forall m \in R(m_0)$, $m_0 \in R(m)$.

For a bounded Petri net, checking whether a marking m is a home marking can be done by determining whether that there exists a non empty path from every marking in the reachability graph to m. Checking whether a net is reversible is equivalent to checking whether the reachability graph is a strongly connected graph, which can be effectively done by Tarjan's Algorithm [126].

For example, the graph of Figure 5.4 has one strongly connected component, which implies that the initial marking m_0 is reachable from any other marking, and thus m_0 is a home marking, and that the P/T net from Figure 5.2 is reversible.

For unbounded P/T nets, the coverability graph does not constitute a tool for determining the reversibility and home markings. The reasons for that have been mentioned earlier and can be reduced to

the facts that he coverability graph does not solve the reachability and path problems(see Section 5.3.6).

5.3.9 Transition Liveness and Quasi-Liveness

For a Petri net representing a system, it is legitimate to ask whether a given transition in the model could ever fire. If we view transitions as actions taken by some agent, transition liveness will tell us whether the agent will ever have a chance to trigger a given action.

Informally, a transition t is said to be quasi-live if it possible to reach a marking where t can fire. A transition t is said to be live if from any reachable marking, it is possible to reach another marking where t can fire. Definition 5.10 gives a formal definition of transition liveness and quasi-liveness.

Definition 5.10 (Liveness)**.**
Let m_0 be the initial marking of a net N, $R(m_0)$ the set of reachable markings and $L(m)$ the set of transition sequences from a marking m.

- *A transition t is Quasi-Live iff: $t \in L(m_0)$.*
- *A transition t is Dead iff: $t \notin L(m_0)$.*
- *A transition t is Live iff: $\forall m \in R(m_0)$, $t \in L(m)$.*

For bounded Petri nets, quasi-live and dead transitions can be verified by inspecting the reachability graph. For a given transition t, if no edge labelled with t can be found in the reachability graph we say that t is dead, otherwise t is quasi-live.

A transition t is live if it is in a firing sequence from every reachable marking. This can be done by considering each marking m of the reachability graph, and starting breath-first search for a marking which has t as an edge. Repeating this process for each transition of the net will answer whether the net is live or not. Note that transition liveness and net liveness are costly properties to verify, because for each transition a search has to be initiated for every reachable marking.

Figure 5.9 illustrates liveness, quasi-liveness and dead transitions. In the reachability graph of Figure 5.9, we can see that t_3 does not appear as an edge in any of the reachable markings, so it is a dead transition. The transition t_4 can not be fired anymore if either of

the markings m_2 or m_3 have been reached, t_4 is thus quasi-live only. Transitions t_1 and t_2 are live, because from every reachable marking it is possible to move to a marking where t_1 and t_2 can fire. When

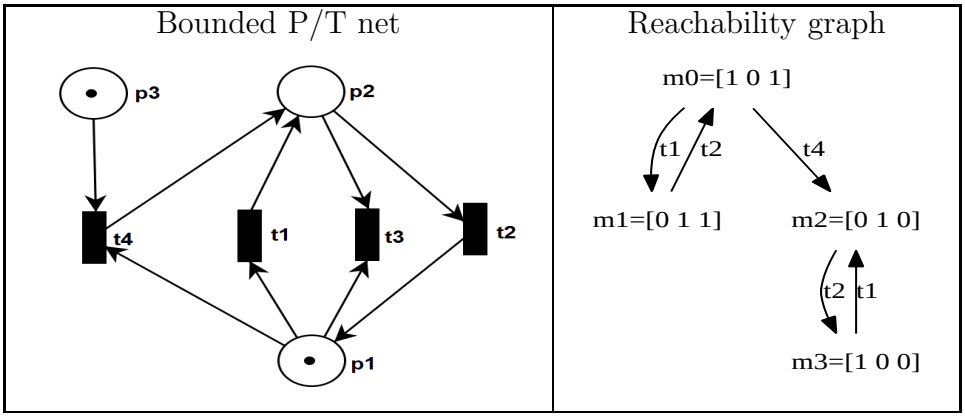


Figure 5.9: Transition liveness and quasi-liveness

a net is reversible and each of its transitions are quasi-live we can conclude that the net is live. The reason is rather simple: When a net is reversible it means that from any marking it is possible to find a sequence that leads to the initial marking. The quasi-liveness of each transition means that from the initial marking all the transitions can eventually fire. We can therefore conclude that each transition can always fire again from any reachable marking, as it is sufficient to go back to the initial marking (reversibility), and further to a marking where the transition in question can fire (quasi-liveness).
For unbounded P/T nets, the quasi-live and dead transitions can be checked in a similar way as with the bounded P/T net. That is, a transition t is quasi-live if it appears at least once in some edge of the coverability graph, otherwise t is dead.

However, the transition liveness and the net liveness cannot be decided by means of the coverability graph, for the same reason as for the marking reachability problem, due to the information hidden by the symbol ω.

5.4 P/T Nets Extended With Inhibitor Arcs

We have so far presented the P/T net class and shown how it can be analysed by means of the reachability or coverability graphs. We shall now give a short introduction to P/T nets extended with inhibitor arcs (PTI). We will provide both informal and formal definitions of PTI nets, and discuss their strength and limitations.

5.4.1 Definitions

P/T nets with inhibitor arcs (PTI) are like the P/T nets presented so far, with one additional element called inhibitor arc.

Inhibitor arcs are types of arcs that can only connect places to transitions. The places from which inhibitor arcs depart are called inhibiting places, and the transitions to which the arcs lead are called inhibited transitions. The transitions firing rule from Definition 5.2 is modified in such a way that a transition can fire if all its input places contain at least one token, and that all its inhibiting places contain no token. A formal definition is given in Definition 5.11.

Definition 5.11 (Definition of PTI nets)**.**
A PTI net is a tuple $PTI = (PT, I)$ *where:*

- PT *is P/T net from Definition 5.1.*
- $I \subseteq (P \times T)$ *is a set of inhibitor arcs, where a pair* $(p, t) \in I$ *describes an inhibitor arc from* p *to* t*.*
- $^\circ t$ *denotes the set of places inhibiting a transition* t*.*
- p°*, denotes the set of transitions inhibited by a places* p*.*
- *A transition* t *can fire in* m *iff:* $\forall p \in^\bullet t, m(p) > 0$ *and* $\forall p \in^\circ t$ $m(p) = 0$*.*

Figure 5.10 shows an unbounded PTI net. Using Definition 5.11, we have $^\circ t_1 = \{p_2\}$, $^\circ t_3 = \{p_1\}$, $p_1^\circ = \{t_3\}$, and $p_2^\circ = \{t_1\}$. Meaning that t_1 can only fire if p_2 has no tokens, and t_3 can only fire if p_1 has no tokens.

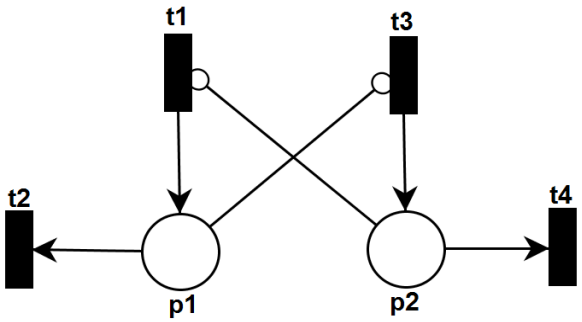


Figure 5.10: An unbounded P/T net with inhibitor arcs

5.4.2 Analysis

The added value by extending P/T nets with inhibitor arcs is the ability to test for zero, which was lacking in P/T nets. With inhibitor arcs, it becomes possible to model Turing machines [62]. Unfortunately, this result also means that for unbounded PTI nets almost no property can be decided any more.

Obviously, for bounded nets all the analysis techniques that were presented in Section 5.3 still hold. When a given PTI net is known bounded, it is possible to generate a reachability graph using Algorithm 1, and study the net's properties by means of the generated graph. In other words, Algorithm 1 is still applicable to bounded PTI nets.

A PTI net can be bounded, if we associate to places a maximum number of token, called place capacity [99]. In [117, 101] we used PTI nets with capacity places to model the dynamics of a drilling control system. Unfortunately, this approach leads to a lost precision, and quickly suffers from the state explosion problem [130].

For unbounded PTI nets, Algorithm 2 which generates a coverability graph does not apply any more. We are thus left without the coverability analysis tools.

The reason that makes Algorithm 2 fails to generate the coverability graph is the lost monotonicity property which binds the marking ordering with transition firing. The monotonicity of P/T net is such that if a transition can fire from a given marking it can also fire from

any larger marking. For PTI nets this property is no longer true.

For example, consider the PTI of Figure 5.10, from the initial marking $m_0 = [0\ 0]$, both t_1 and t_3 can fire. Despite the fact that triggering t_1 leads to a larger marking $m_1 = [1\ 0]$, t_3 can not fire from m_1, because of the inhibitor arc from p_1 to t_3.

Algorithm 2 exploits the monotonicity property of P/T nets when inserting the ω in the acceleration function. Inspired by this idea, we shall in the next chapter define a subclass of PTI nets on which it is possible generate a coverability graph and study the model's properties. For instance, the PTI net of Figure 5.10 belongs to such a subclass, and models a situation that P/T nets could not have modelled. This will be studied and discussed in the next chapter.

5.5 Chapter Summary

This chapter has introduced the P/T net class, its analysis by means of state space exploration, and its extension with inhibitor arcs.

In computer aided model checking, a system designer proposes a model, and the computer checks whether the model satisfies the defined properties. We have considered the following properties: boundedness, deadlock, marking reachability, home marking, reversibility, quasi-liveness and liveness.

For any kind of bounded Petri net, even beyond the P/T net class, the reachability graph represents the complete state space of the model, and can thus be used to decide all the above mentioned properties. Generating the reachability graph can be done by Algorithm 1.

When a given Petri net is unbounded, Algorithm 1 does not terminate, meaning that a reachability graph can only be generated when we a priori know that the net is bounded. Since boundedness is also a property that we wish to determine, we need an alternative approach.

An alternative solution is to generate a coverability graph using Algorithm 2, under the condition that the net in question belongs the P/T net class. This algorithm generates a reachability graph if the net in question is bounded, and a coverability graph if it is unbounded.

Unfortunately, it runs slower, it is limited to P/T nets only, and not all properties of interest can be decided by it.

Table 5.2 summarises which properties can be decided by means of the state space analysis. All these properties are of interest for the next chapter in which we shall present a subclass of PTI nets on which these properties are decidable by coverability graph analysis.

Table 5.2: Deciding P/T net properties by means of reachability or coverability graphs

Problems	**Unbounded (Coverability)**	**Bounded (Reachability)**
Boundedness	Decidable	Undecidable
Deadlock	Decidable	Decidable
Marking reachability	Undecidable	Decidable
Sub marking reachability	Undecidable	Decidable
Path	Undecidable	Decidable
Home making	Undecidable	Decidable
Reversibility	Undecidable	Decidable
Quasi liveness	Decidable	Decidable
Liveness	Undecidable	Decidable

5.6 Algorithms

Algorithm 1 Finding the Reachability graph of a P/T net

1: Initialization:
2: $(V, E, v_0) \leftarrow (m_0, \emptyset, m_0)$ {*Initialize graph with m_0 as its only vertex*}
3: **function** $ReachGraph(V, E, v_0)$
4: $S : set \leftarrow \{m_0\}$ {*S is a set filled with m_0 at start*}
5: **while** $S \neq \emptyset$
6: *select* $m \in S$ {*Choose an entry m from S*}
7: $S \leftarrow S \backslash \{m\}$ {*Remove the entry m from S*}
8: **for all** $t \in enabled(m)$
9: $m' \leftarrow fire(t, m)$ { *enable and fire from Def. 5.2*}
10: **if** $m' \notin V$
11: $V \leftarrow V \cup \{m'\}$ {*Add the next marking to the set of vertices*}
12: $S \leftarrow S \cup \{m'\}$ {*Add the next marking to the S*}
13: $E \leftarrow E \cup \{(m, t, m')\}$ {*Connect m and m' with an arc t *}
14: **return** (V, E, v_0) {*Return the Reachability graph*}

Algorithm 2 Finding the Coverability graph of a P/T net

1: Initialization:
2: $(V, E, v_0) \leftarrow (m_0, \emptyset, m_0)$ {*Initialize graph with* m_0 *as its only vertex*}
3: **function** $CovGraph(V, E, v_0)$
4: $S : set \leftarrow m_0$ {S *is a set filled with* m_0 *at start*}
5: **while** $S \neq \emptyset$
6: *select* $m \in S$ {*Choose an entry* m *from* S}
7: $S \leftarrow S \backslash \{m\}$ {*Remove the entry* m *from* S}
8: **for all** $t \in enabled(m)$
9: $m' \leftarrow fire(t, m)$ { *enable and fire from Def. 5.2*}
10: $m' \leftarrow Accelerate(m, t, m', V, E)$
11: **if** $m' \notin V$
12: $V \leftarrow V \cup \{m'\}$ {*Add* m' *to the set of vertices*}
13: $S \leftarrow S \cup \{m'\}$ {*Add the next marking to* S}
14: $E \leftarrow E \cup \{(m, t, m')\}$ {*Connect* m *and* m' *with an arc* t }
15: **return** (V, E, v_0) {*Return the Coverability graph*}
16: **function** $Accelerate(m, t, m', V, E)$
17: **for all** $m'' \in V$
18: **if** $m'' < m'$ and $m'' \in m_0[\sigma\rangle m$
19: $m' \leftarrow m' + (m' - m'') \times \omega$ {*Where* $+, -$, *and* $\times$ *are vector addition, vector substraction, and scalar multiplication respectively*}
{ *If* m' *covers* m'' *(see Def 5.4) and* m'' *appears in the path from* m_0 *to* m*, then insert* ω *in all places* p $m'(p) > m''(p)$ }
20: **return** m'

Chapter 6

Place/Transition nets with Inhibitor Arcs

Place/Transition nets (P/T nets) have shown to be a very useful modelling tool, but they sometimes fail to model relatively simple situations. For this reason, several extensions have been proposed to improve their modelling power. Among these extensions we find P/T nets extended with inhibitor arcs (PTI). The basic capability that inhibitor arcs add to P/T nets is referred to as zero testing.

The problem caused by this extension was elegantly expressed by James L. Peterson [107]:

> *In general, it seems that any extension which does not allow zero testing will not actually increase the modelling power (or decrease the decision power) of Petri nets but merely result in another equivalent formulation of the basic Petri net model. (Modelling convenience may be increased.) At the same time, any extension which does allow zero testing will increase the modelling power to the level of Turing machines and decrease decision power to zero. Thus Petri net extensions would seem to have few practical advantages for analysis.*

This chapter discusses the poor decision power of PTI nets, based on which a sub-class of PTI nets is presented. We call this class Cohesive Place/Transition nets with Inhibitor Arcs (CPTI), which

is such that inhibitor arcs are used in a restrictive way. This class is then further restricted to a class called Mutually Inhibited Cohesive Place/Transition nets with Inhibitor Arcs (MICPTI). We show how the marking reachability, deadlock, reversibility, home marking, quasi-liveness, liveness and path problems are all decidable on MICPTI nets.

6.1 Introduction

Place/Transition nets with inhibitor arcs [4] (PTI) are extension to P/T nets, and have already been introduced in Chapter 5. The very fundamental ability that is added by inhibitor arcs is that an action is not only conditioned by the presence of resources but could also be conditioned by their absence.

Intuitively, in a P/T net, the presence of more tokens implies the possibility of firing more transitions, because transition firings are conditioned by the presence of tokens in some given places. This property does not hold for PTI nets, because the presence of tokens could also imply transition inhibiting rather than enabling. The property that is lost by introducing inhibitor arcs is called the monotonicity property of P/T nets [48, 53, 47, 5, 38], which to the best of our knowledge, is what makes PTI nets so hard to analyse.

The PTI nets class was proven Turing Powerful [61, 44], thus introducing inhibitory arcs leads to a formalism that can model any system. A consequence of the Turing power of PTI nets, is that the general coverability graph procedure [84] does not apply to PTI nets (it will be discussed in Section 6.3), because if that were the case, the termination problem would be decidable [61]. However, it is a well-known fact that the termination problem is undecidable on a Turing machine [129].

Despite the undecidability results of PTI nets, it remains possible to find some sub-classes which can be analysed. In other words, it seems possible to use inhibitor arcs in some restricted forms and still have a high *Decision Power*. An example of such a class is the primitive systems [23], also presented in Section 6.3. Following a similar direction as with primitive systems, we propose another

sub-class of PTI called Cohesive PTI (Section 6.4), in which it is possible to compute a coverability graph (Section 6.6). The Cohesive PTI class is then further restricted to what we call Mutually Inhibited Cohesive PTI sub-class. The advantage of the latter is that its coverability graph embeds sufficient information on the behaviour of the net, so that it becomes possible to decide: Marking reachability, deadlock, reversibility, home marking, quasi-liveness, liveness and path problems (see Section 6.8).

But before we proceed to the next section, we shall emphasize the need of using inhibitor arcs based on a concrete example from drilling.

Figure 6.1 shows a PTI net capturing a simple part of the rig dynamics. The transitions t_1 and t_2 are responsible for increasing and decreasing the drill-string rotational speed respectively. The transitions t_5 and t_6 are responsible for increasing and decreasing the drill-string downward speed. While t_3 and t_4 are responsible for releasing and activating the power-slips. The rotational and downward speeds can only be set if the power-slips are released. Likewise the power-slips can only be activated if neither the rotational nor the downward speeds are set. These constraints can unfortunately not be captured without using inhibitor arcs.

The use of inhibitor arcs means that the proposed model cannot be analysed within the actual Petri net theory. We shall therefore suggest a sub-class of PTI nets, that satisfies our modelling needs, and that is fully analysable.

6.2 Turing Equivalence

This section presents the Turing equivalence proof of PTI nets which was first formulated in [61], and later on refined in [107]. Since we shall use inhibitor arcs, we found it logical to show why they are fundamentally so problematic. We do that by reproducing the Turing equivalence proof as presented in [107].

The proof of Turing equivalence is based on the registry machine of Shepherdson in [123]. A registry machine is an abstract machine which has registers for storing arbitrarily large values. The main result of [123] is that a registry machine which has the following

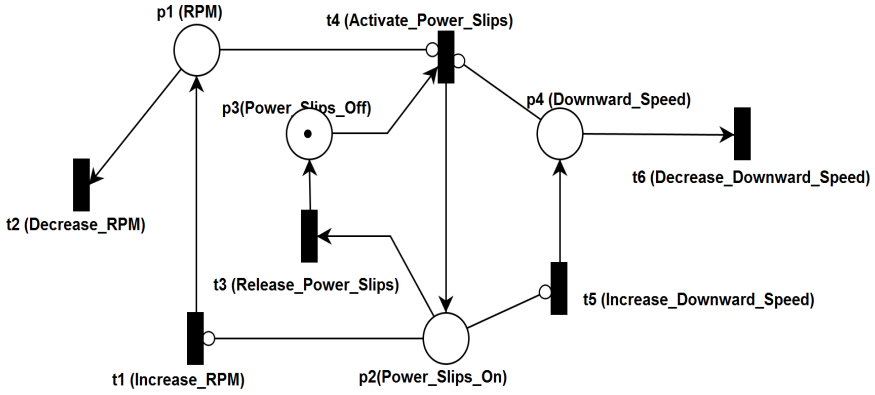
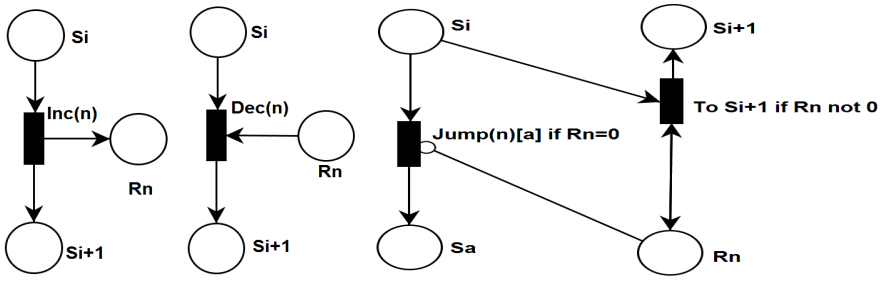


Figure 6.1: A PTI net modelling parts of a rig dynamics

instruction is Turing equivalent:

- $Inc(n)$: Increment register n by 1.
- $Dec(n)$: Decrement register n by 1, only if $n \neq 0$.
- $Jump(n)(a)$: Jump to instruction a if register $n = 0$ otherwise move to the next instruction.



(a) $Inc(n)$ (b) $Dec(n)$ (c) $Jump(n)(a)$

Figure 6.2: A representation of register machine instruction set using PTI

These instructions can be represented using PTI as shown in Figure 6.2. To represent a program with the above instructions using PTI we use n places to represent the set of registers $Reg = \{R_1, R_2, ..., R_n\}$. For a program with k statements we use $k+1$ places to represent the

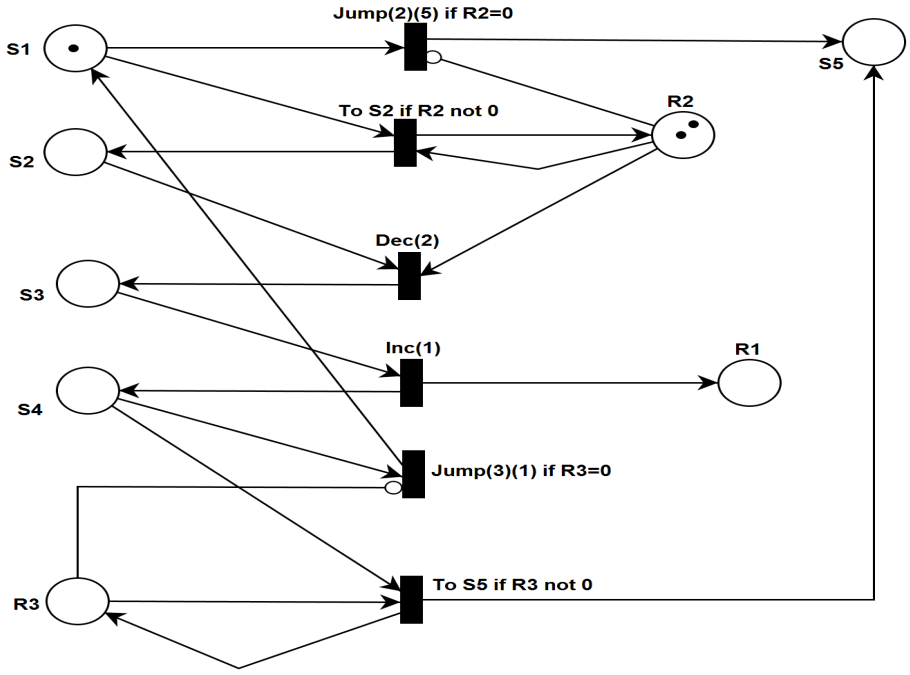


Figure 6.3: A PTI net representing a program that adds R_2 to R_1, with initial values $R_2 = 2$ and $R_1 = 0$.

statement's position in the program, $S = \{s_1, s_2, ..., s_{k+1}\}$. Finally, each instruction in the program is represented by a transition. For example, Figure 6.3 shows a program that adds the content of register R_2 to register R_1 (it also uses an always zero register R_3) using the instructions of Algorithm 3. The point is that PTI net can

Algorithm 3 This program adds the content of register R_2 to R_1

1: S_1: $Jump(2)(5)$	{*If* $R_2 = 0$ *go to* S_5}
2: S_2: $Dec(2)$	{*Substract 1 from* R_2 *and move to* S_3}
3: S_3: $Inc(1)$	{*Add 1 to* R_1 *and move to* S_4}
4: S_4: $Jump(3)(1)$	{*Go to instruction* S_1}
5: S_5: Halt	

simulate Shepherdson's registry machine [123], and are therefore

Turing equivalent. Obviously, since problems like boundedness and marking reachability can be used to determine program termination, they must be undecidable for PTI nets.

6.3 Coverability Graph Problem

The Turing equivalence of PTI nets provides a general proof that no general coverability graph procedure exists which can apply to any PTI model. But, the proof does not give precise reasons as to which properties are lost or introduced so that the analysis of PTI becomes undecidable. This section investigates these reasons.

Since we are dealing with potentially infinite systems (unbounded places), the reachability graph analysis does obviously not provide a solution, because the procedure generating the reachability graph will not terminate, see Section 5.3. We will therefore focus on the generation of the coverability graph, a procedure which is directly derived from the Karp and Miller Tree [84].

So, to find an explanation for why the coverability graph procedure does not apply to PTI nets, we could start by finding out why the algorithm applies to P/T nets. This question was studied in [5, 47, 38, 22] through the so-called Well Structured Transition Systems, or WSTS for short.

WSTS define a general structure which was introduced mainly to generalize the fundamental concepts behind the construction of the coverability graph. From the perspective of WSTS, P/T transition nets are transition systems which are equipped by an ordering relation $\leq$ over the set of reachable markings, and that the relation $\leq$ is *compatible* with $\rightarrow$ (the transition relation between two markings). The *compatibility* between $\leq$ and $\rightarrow$ is the so-called *monotonicity property*. In the case of P/T nets we have a stronger form of compatibility called *strict compatibility*. Note that *strict compatibility* is such that, if $m_1 < m_2$ and $m_1 \rightarrow^t m_3$ then there exist $m_2 \rightarrow^t m_4$ such that $m_3 < m_4$.

In contrast to P/T nets, PTI nets do not satisfy the *compatibility* requirement, that is why Algorithm 2 (see Section 5.3) which generates a coverability graph P/T nets can not be applied to PTI.

Recall that in Algorithm 2, the *monotonicity property* is actually exploited to decide the insertion of ω (Definition 5.4). Deciding when to introduce the ω symbol is handled by the acceleration function.

Roughly, the coverability graph procedure works as follows: The algorithm starts from an initial marking m_0, and proceeds by determining the possible successor markings. Each successor marking is run through the acceleration procedure, for insertion or not of the symbol ω. A consequence of the *monotonicity property* is that if $m_1 < m_2$ and their exists a non empty sequence of transitions σ from m_1 to m_2, then it is possible to repeat σ from m_2 and reach an even higher marking. When a marking has passed through the acceleration function, it is either added to the graph (if not in the graph already) or ignored. The procedure is repeated for every added marking in a similar way as with m_0.

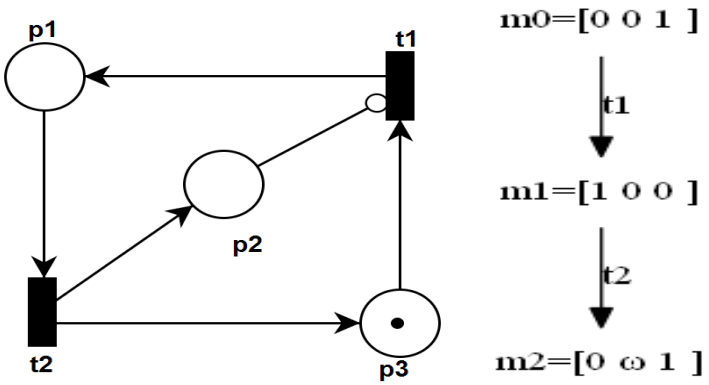
Now, because the *monotonicity property* is lost for PTI nets, the acceleration procedure does not work correctly any more. It is incorrect in two possible ways:

1. The symbol ω is inserted for a place which is actually bounded.
2. The symbol ω is inserted too early, and the resulting graph is not a coverability graph anymore, i.e not every reachable marking is coverable

Figures 6.4 and 6.5 illustrate the above cases respectively. Consider the PTI net in Figure 6.4a and its corresponding coverability graph in Figure 6.4b, starting from $m_0 = [0\ 0\ 1]$. The transition t_1 is the only enabled transition, removing 1 token from p_3 and adding 1 token to p_1, we get a marking $m_1 = [1\ 0\ 0]$. Since m_0 is not comparable to m_1 the acceleration procedure does not insert any ω. Then, t_1 can not fire since p_3 has no tokens, but t_2 can fire, putting 1 token in each of p_2 and p_3 leading to a marking $m_2' = [0\ 1\ 1]$ from which no transition can fire any more. However, due to the fact that $m_0 < m_2'$ the acceleration procedure generates $m_2 = [0\ \omega\ 1]$ which is clearly wrong, since it indicates that p_2 is unbounded while it is bounded.

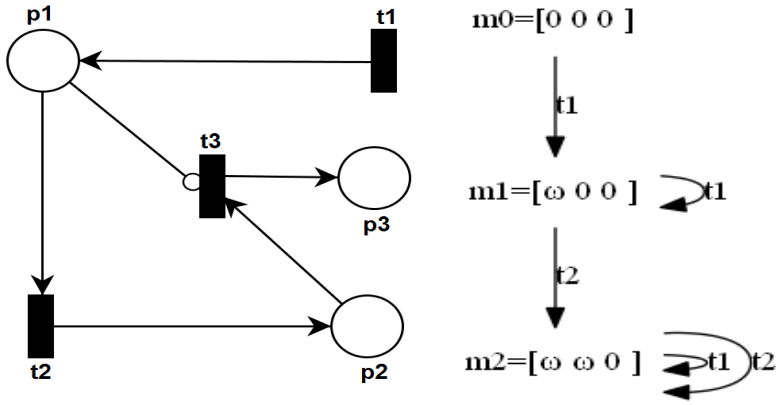
Figure 6.5a and Figure 6.5b show the case where ω is inserted too fast in m_1 and m_2, resulting in a coverability graph where p_3 is never filled with a token. However, from the initial marking m_0 it is possible to repeatedly fire $\langle t_1, t_2, t_3 \rangle$ and actually have an arbitrary large number of tokens in p_3. This case shows that not all reachable

markings of the PTI in 6.5a are covered by the coverability graph in 6.5b.



(a) (b)

Figure 6.4: The place p_2 in 6.4a is bounded, but m_2 in 6.4b has ω



(a) (b)

Figure 6.5: In 6.5a p_3 is actually unbounded (repeating $\langle t_1, t_2, t_3 \rangle$), but p_3 is always 0 in the coverability graph 6.4b

However, a subclass of PTI nets called Primitive systems, introduced by Busi in [23], shows that it is possible to use inhibitory arcs

in some restricted manner and still produce nets with high decision power.

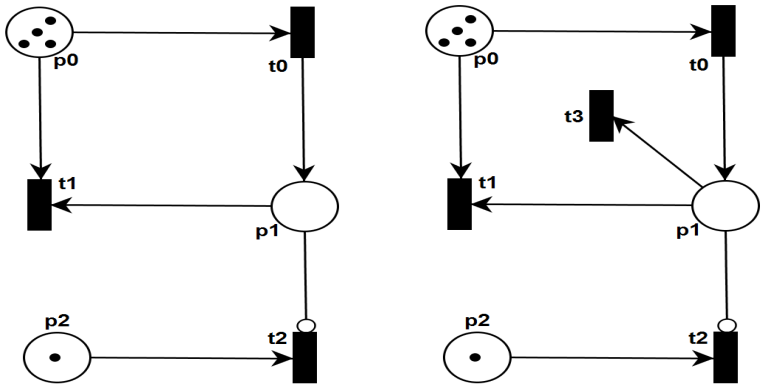
Primitive systems [23] are a subclass of P/T nets with inhibitory arcs for which it is possible to construct a coverability graph, and thus properties like boundedness of net, boundedness of a place, and quasi-liveness of a transition become solvable. Primitive systems handle the lost *monotonicity property* by imposing one constraint on the net construction. In Primitive Systems an emptiness limit is associated to each inhibiting place, in such a way that whenever this limit is exceeded the corresponding place can not be emptied any more.

Definition 6.1 (Definition of Primitive Systems)**.**
A PTI net N is Primitive if we can compute the emptiness limit EL such that: $\forall p \in I\ \forall m \in R(m_0)$, if $m(p) > EL(p)$ then $\forall m' \in R(m)$, $m'(p) \neq 0$.

Figure 6.6a shows a Primitive system where the place p_1 can not be emptied after its number of tokens exceeds 2. For clarity, from the initial marking, if we fire t_0 three times to deposit 3 tokens in p_1, then t_1 can only fire once to remove one token from both p_0 and p_1 after which its not possible to fire neither t_0 nor t_1, and thus p_1 can not be emptied any more. So, when the number of tokens in p_1 exceeds 2 it is not possible to empty p_1 any more. Adding a transition t_3 as shown in Figure 6.6b makes the net non Primitive, because as long as there are tokens in p_1 these can be removed by t_3. In this work we propose another class of PTI nets called Cohesive PTI. Our hypothesis is that it should be possible to determine the coverability graph if we know how to insert the ω symbol in a correct manner, Cohesive PTI nets offer this possibility, as demonstrated next.

6.4 Cohesive Place/Transition Nets with Inhibitors

Cohesive Place/Transition Nets with Inhibitor arcs (CPTI) is first of all a PTI net as defined earlier in Definition 5.11 with additional



(a) A Primitive system with EL(P1)=2, redrawn from [23]

(b) Non Primitive system

Figure 6.6: An example of primitive system nets

model construction constrains. This section defines the CPTI class of nets.

In a CPTI net only two structures are allowed. The first one represents on/off type of actions, and is defined as a circular elementary structure (*ces*). The other one represents level type of actions, and is a flat elementary structure (fes). Both *ces* and fes structures are defined in Definition 6.2. For example, turning on or off the mud pump are on/off type of actions, while increasing or decreasing the mud flow-rate are level type of actions. These two elementary structures are shown in Figure 6.7.

Roughly speaking in fes type of structure only one place is involved. It has exactly one input transition t_1, one output transition t_2, and $t_1 \neq t_2$. In addition the input transition t_1 must have no input places, and the output transition t_2 must have no output places. Figure 6.7a shows a flat elementary structure as defined by Definition 6.2.

In a *ces* type of structure, exactly two places and two transitions are involved. These are structured in a token conserving manner as stated by Definition 6.2 and shown in Figure 6.7b.

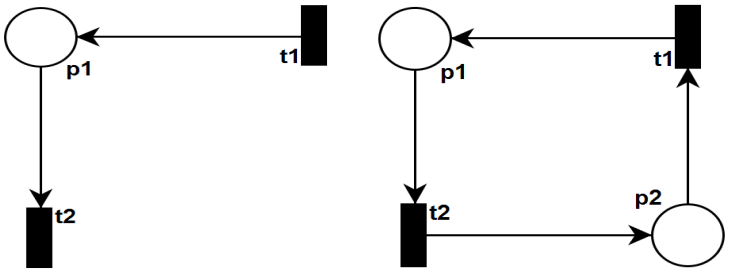
Definition 6.2 (Elementary Structures)**.**

Let places $p, p' \in P$ and $t, t' \in T$, and let $cond_a$ $cond_b$ and $cond_c$ be three conditions where:

- $cond_a = ({}^\bullet p = \{t\}) \wedge (p^\bullet = \{t'\}) \wedge (t \neq t') \wedge (t^\bullet =^\bullet t' = \{p\})$.
- $cond_b = t'^\bullet =^\bullet t = \emptyset$.
- $cond_c = (t'^\bullet =^\bullet t = \{p'\}) \wedge ({}^\bullet p' = \{t'\}) \wedge (p'^\bullet = \{t\})$.

A triple (p, t, t') is fes iff: $cond_a$ and $cond_b$ hold, denoted $cond_{fes}$.

A quadruple is (p, p', t, t') is ces iff: $cond_a$ and $cond_c$ hold, denoted $cond_{ces}$.



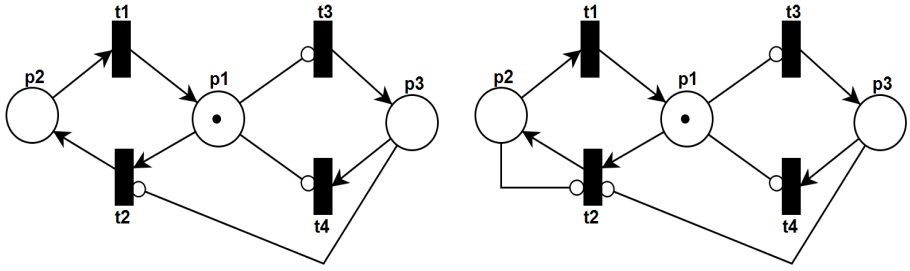
(a) Flat elementary structure (b) Circular elementary structure

Figure 6.7: In fes only one place is used, while *ces* involves exactly two places and two transitions in a token conserving manner

Furthermore, in a CPTI net the use of inhibitor arcs is only allowed across elementary structures. That is, it should not be possible to use inhibitor arcs between places and transitions belonging to the same elementary structure. Figure 6.8 shows two nets, the one from Figure 6.8a is CPTI while the other from Figure 6.8b is not because of the inhibitor arc linking p_2 to t_2. This concept is formally defined in Definition 6.3.

Definition 6.3 (Definition of CPTI nets)**.**
Let $ES = \{es_1, ..., es_m\}$ be the set of elementary structures of a net N. Let T_i and P_i be the sets of transitions and places of es_i respectively, where:

- $\forall p \in P, \exists es_i \in ES$ *such that* $p \in P_i$.
- $\forall t \in T, \exists es_i \in ES$ *such that* $t \in T_i$.



(a) A Cohesive PTI (b) A None Cohesive PTI

Figure 6.8: The net in Fig 6.8a is a CPTI, while the net in Fig 6.8b is not, because of the inhibitor arc from p_2 to t_2. Inhibitor arcs within the same elementary structure are not allowed.

The net N is a CPTI net iff:

– $\forall es_i \in ES\ \forall p \in P_i$ *either* $p^\circ = \emptyset$ *or* $\forall t \in p^\circ, t \notin T_i$.

Algorithm 4 (Section 6.13) takes as input a PTI net and determines whether the net is a CPTI net or not. The algorithm extracts all the elementary structures and stores them in the variable ES, when a structure that is neither *ces* nor fes is found, the algorithm returns $false$. Once all the elementary structures are determined, it checks whether inhibitor arcs violate Definition 6.3; *i.e.* no transition should be inhibited by a place contained in the same structure.

6.5 Monotonicity of Cohesive PTI Nets

CPTI nets have already been defined in Section 6.4. This section presents the monotonicity property of CPTI nets, which basically defines the fundamental behaviour of CPTI nets. Since the monotonicity of P/T nets is what allows the generation of coverability graphs, we shall now identify what kind of monotonicity property holds for CPTI nets. This property will be exploited later when we propose an algorithm that generates a coverability graph for CPTI nets.

The general behaviour of CPTI is dictated by its two basic structures (*ces* and fes). From a given marking, firing a transition

can give rise to a new marking in a limited and deterministic manner. Intuitively, when firing a transition t in a fes structure, it either adds one token to a place or removes one token from one place. A transition t which is in a *ces* structure, removes one token from a place and adds one token to another place. In either cases, the marking that emerges from firing t can be anticipated based on what structure the transition belongs to, as expressed by Property 6.1.

Property 6.1 (CPTI Dynamics)**.**
Let N be a CPTI net. From any marking m, any enabled transition t at m generates a new marking m' in one the following ways:

1. *$m' > m$, and m' equals m for all places except one; i.e. $\exists! p \in P$ such that $\forall p_i \neq p \in P$ $m'(p_i) = m(p_i)$ and $m'(p) > m(p)$.*
2. *$m' < m$, and m' equals m for all places except one; i.e. $\exists! p \in P$ such that $\forall p_i \neq p \in P$ $m'(p_i) = m(p_i)$ and $m'(p) < m(p)$.*
3. *m' and m are not comparable; i.e. $\exists p_a, p_b \in P$ where $m'(p_a) < m(p_a)$ and $m'(p_b) > m(p_b)$.*

NB! See proof 6.1 in Section 6.12.

The introduction of inhibitory arcs causes the loss of the strict monotonicity property in P/T nets (strict ordering is not compatible with the transition relations). Recall that the strict monotonicity in Petri net is that; if $m_1 < m_2$ and $m_1 \rightarrow^t m_3$ then there exist $m_2 \rightarrow^t m_4$ such that $m_3 < m_4$; *i.e.* if a transition could fire from a marking it can fire at any higher marking. This property is not satisfied for CPTI.

Property 6.2.
CPTI nets are not strictly monotone.
NB! See proof 6.2 in Section 6.12.

The kind of monotonicity that is satisfied by CPTI is the following: From a given m if by firing a transition t we can reach a marking $m' > m$, then it is possible to fire t from m', we call this for *T-monotonicity*. In contrast to strict monotonicity, t can not fire from any $m' > m$, but only from those markings that are generated by t.

Definition 6.4 (Definition of T-monotonicity)**.**
T-monotonicity is such that; if $m_1 < m_2$ and $m_1 \rightarrow^t m_2$ then $\exists m_3$, such that $m_2 \rightarrow^t m_3$ and $m_2 < m_3$.

Property 6.3.
Any CPTI net satisfies the T-monotonicity.
NB! See proof 6.3 in Section 6.12.

6.6 Cohesive PTI Coverability

In this section we show how to compute the coverability graph for a CPTI net. Recall that in Section 6.3 we argued that the lost *monotonicity property* of PTI had the consequence of either a too early or an incorrect insertion of ω. The idea behind CPTI nets is that there is no ambiguity in determining whether a place can contain arbitrarily many tokens or not. In other words, CPTI nets enable a correct insertion of ω.

A correct insertion of ω is done by taking advantage of Properties 6.1 and 6.3. This means that the constrained behaviour of CPTI nets combined with their *T-monotonicity* allow a correct insertion of ω. Algorithm 5 (Section 6.13) works very much like the one in Algorithm 2, with a slightly different acceleration function.

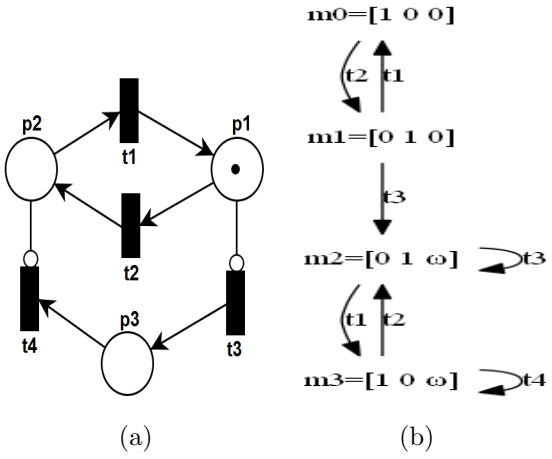
Intuitively, when generating the reachability graph, that is an exhaustive enumeration of all the possible markings, for each marking m and its successor m', we meet one of the cases of Property 6.1. The elementary structure in *Case 2* is fes, because only one place is involved, combined with the fact that $m'(p) < m(p)$ gives no reason of acceleration (ω insertion). The resulting m' has either been visited earlier or will be considered as a new marking.

In *Case 3* we are faced with ces, and by construction m' is not comparable to m. The acceleration can only take effect in *Case 1* when operating on a fes. That is, from a marking m a transition t has added a token in place p, based on the *T-monotonicity* transition t can fire again.

Knowing that an inhibitor can not exist inside a given elementary structure, and knowing that t can fill exactly one place, we can conclude that t can fire again from m' because an added token can simply not inhibit t from firing again (but can inhibit other transitions than t) leading to the successor marking of m' by t. Thus, the transition t can fire arbitrarily many times and ω can safely be

inserted.

In Algorithm 5 the ω insertion is done in the acceleration function. Figure 6.9 shows a CPTI net and its corresponding coverability. Applying Algorithm 5 to the CPTI net of Figure 6.9 does the following: It starts from the initial marking m_0 where only t_2 can fire leading to m_1. Because m_0 and m_1 are not comparable, no acceleration takes effect. From m_1, t_1 can fire to go back to m_0 with no acceleration taking effect either, but t_3 leads to a marking $m_2 = [0\ 1\ 1]$. Since $m_2 > m_1$, the acceleration takes effect causing $m_2(p_3) = \omega$. From m_2 we can fire t_3 and t_4, because $\omega + 1 = \omega$ (see Definition 5.4) no new node is generated by t_3, but firing t_1 leads to m_3 from which it is possible to fire t_2 and go back to m_2 or fire t_4 and no node is generated($\omega - 1 = \omega$). Note that under the execution of Algorithm 5, if a generated node already exists it will be ignored. To prove the finiteness and correctness of Algorithm 5 on CPTI nets,



(a) (b)

Figure 6.9: A CPTI net and its corresponding coverability graph

and inspired from [84, 23, 107] we use two lemmas: Koenig [86] and a reformulation of Dickson's lemma in the context of P/T nets [36].

Lemma 6.1 (Koenig's Lemma)**.**
If G is a connected graph with infinitely many vertices such that every vertex has finite degree then G contains an infinitely long simple path;

i.e. a path with no repeated vertices. Consequently, if G is such that every vertex has finite degree and G contains no infinite simple path, than G is finite.

Lemma 6.2 (Dickson's Lemma Reformulation)**.**
For every infinite sequence of markings, there exists an infinite subsequence that is increasing with respect to the ordering relation $\leq$.

Proposition 6.1 (Finiteness)**.**
Finiteness For a CPTI net, Algorithm 5 computes a finite graph.
NB! See proof 6.4 in Section 6.12.

Corollary 6.1 (Termination)**.**
For a CPTI net, Algorithm 5 terminates.
NB! See proof 6.4 in Section 6.12.

Proposition 6.2 (Correctness)**.**
For a CPTI net Algorithm 5 computes a coverability graph.
NB! See proof 6.5 in Section 6.12.

Corollary 6.2.
Let G be a graph computed by Algorithm 5 for a CPTI net N. Every reachable marking in a N is either explicitly present in a node of G or covered by an extended marking (some node containing the symbol ω*) in G.*
NB! See proof 6.5 in Section 6.12.

Proposition 6.3 (Reachability)**.**
For a CPTI net, a marking m that is covered by an extended marking is not necessary a reachable marking.
NB! See proof 6.6 in Section 6.12

6.7 Analysis of Cohesive PTI

We have so far presented CPTI nets, and proposed an algorithm (Algorithm 5) that generate a coverability graph for them. This section shows which model properties can be decided using the coverability graph computed by Algorithm 5.

We have already gone trough the exercise of deciding P/T nets properties in Section 5.3. This section repeats that exercise for the CPTI class of nets. The properties of interest are the following: boundedness, deadlock, marking and sub-marking reachability, path, home marking, transition liveness and quasi-liveness.

6.7.1 Boundedness

Recall that from Definition 5.3 a net is bounded if none of its places can contain an arbitrarily large number of tokens. Consequently, if one place can contain an arbitrary large number of tokens, the net becomes unbounded.

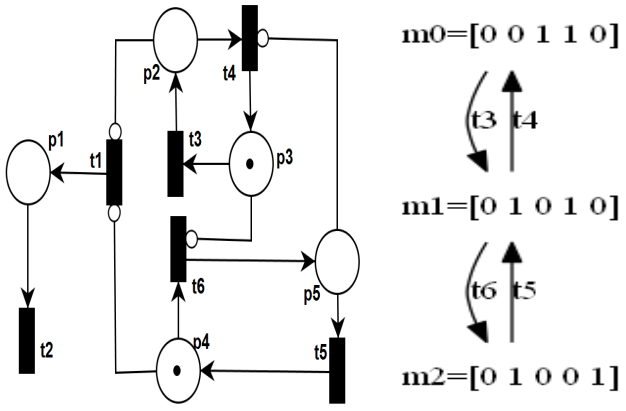
Using the coverability graph, the existence of one extended marking i.e., the presence of ω implies that the net is unbounded. Note that if a CPTI net contains some fes type of structures does not necessarily mean that the net is unbounded. Figure 6.10 shows a bounded CPTI net, since its corresponding coverability graph does not contain any ω symbol, despite the fact that place p_1 and transitions t_1 and t_2 constitute a fes.

On the other hand, Figure 6.11 shows an unbounded CPTI net and its corresponding coverability graph. Notice that the only difference between Figures 6.10a and 6.11a is the inhibitor arc between p_5 and t_4, which is present only in Figure 6.10a. Boundedness is thus decidable for CPTI nets as it is for P/T nets (Section 5.3).

6.7.2 Deadlock

A deadlock marking describes a situation in which no transition can fire any more. From Definition 5.7, a marking of a net N is deadlock, if the net's corresponding reachability (or coverability) graph does not contain any outgoing arcs. This fact also applies for CPTI nets.

Figure 6.12 shows a CPTI net and its corresponding coverability graph. The marking m_3 is a deadlock marking, because it has no outgoing arc from it. The marking m_3 describes in fact infinitely many deadlock situations, which is explained as follows: From m_0, t_2 can fire to remove a token from p_1 and insert one token in p_2. The firing of t_2 leads to the marking m_1, from which it is possible to fire t_3



(a) (b)

Figure 6.10: Bounded CPTI net

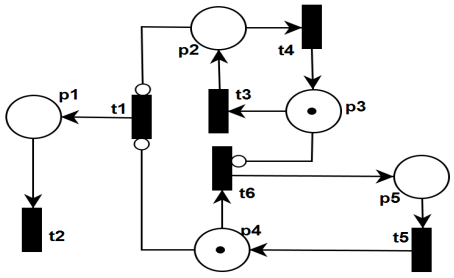
as often as desired, but as soon as t_1 is fired again, no transition can fire any more, reaching a deadlock marking. Deadlock markings are thus decidable for CPTI nets as they are for P/T nets (Section 5.3).

6.7.3 Marking and Sub-Marking Reachability

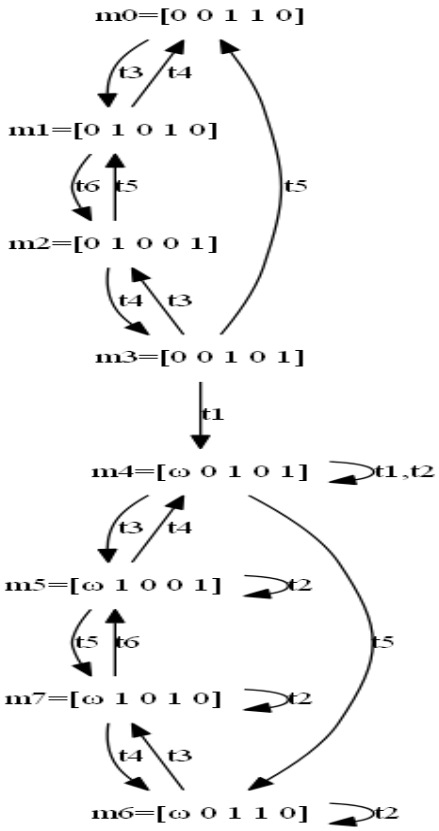
The marking and sub-marking reachability problems were presented and analysed for P/T nets in Section 5.3. This analysis also holds for CPTI nets. That is, for a bounded CPTI net these problems are decidable by inspecting the reachability graph.

For unbounded nets these problems cannot be decided by means of the coverability graph. To explain this undecidebility we refer to Propostions 6.3 and Corollary 6.2, where for any marking m in a coverability graph of a CPTI net the following holds:

1. If m is explicitly present in the coverability graph, then it is a reachable marking. This can be checked by inspecting the graph nodes.
2. If m is covered by some extended marking in the coverability graph, then m can be either reachable or not.
3. If none of the above cases hold then m is not reachable.

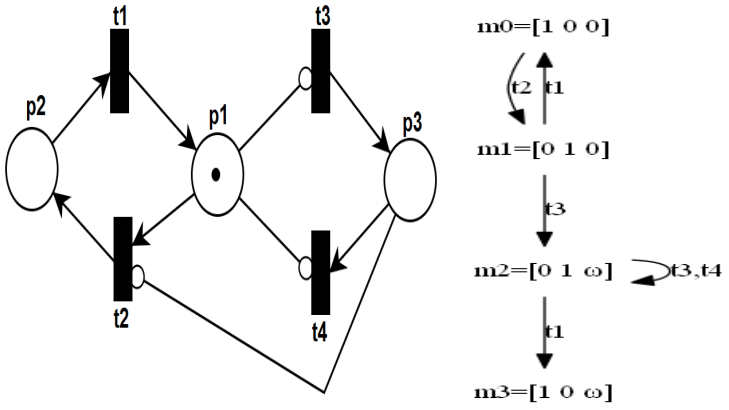


(a)



(b)

Figure 6.11: Unbounded CPTI net



(a) (b)

Figure 6.12: CPTI nets and deadlock markings

NB! This fact constitutes a motivation of Section 6.8 in which a subclass of CPTI nets is introduced, such that the reachability problem becomes decidable.

6.7.4 Path

Intuitively, the problem of determining a path between two reachable markings of a CPTI net is a difficult problem to solve. First, because the marking reachability cannot be solved by means of the coverability graph. Second, even if we were ask to find a path between two marking which are known reachable, the problem remains difficult.

To illustrate this problem we use Figure 6.13, which once again represents a CPTI net with its corresponding coverability graph. Now, consider that we want to find a path between two a priori known reachable markings $m_a = [4\,0\,1\,3]$ and $m_b = [3\,1\,0\,2]$. From the Coverability graph of Figure 6.13b, m_a is covered by m_6 and m_b is covered by m_7. However, finding a path from m_a to m_b can not be reduced to finding a path from m_6 to m_7. From the marking m_a, if we apply t_3 we will reach some marking $m'_a = [4\,1\,0\,3]$ from which we can fire t_2 once and reach $m''_a = [3\,1\,0\,3]$, but to reduce the last coordinate from the value 3 to 2 as specified by m_b is not feasible neither from

m_7 nor from m_6. In fact, in order to decrease the last coordinate we have to be in some extended marking where it is possible to fire t_6.

However, in order to fire t_6 we need to completely empty p_1. One possible path from m_a to m_b could be σ such that:
$\sigma = \langle t_3, t_2^4, t_6, t_4, t_1^3, t_3 \rangle$. Clearly Breadth first search does not make much sense in this case, because the coverability graph masks information about the marking connections. The complexity of the path problem in addition to the reachability problem will be handled in Section 6.8 with the so-called Mutually Inhibited CPTI.

6.7.5 Home Marking and Reversibility

For a given net, determining whether a marking can be reached from any other marking is called a home marking, and when the initial marking is a home marking, then the net becomes a reversible net. These notions have already been introduced in Section 5.3 and Definition 5.9.

The analysis for determining home markings and net reversibility for P/T nets also holds for CPTI nets. That is, a marking m is a home marking if there exists a path from any marking in the reachability graph to m. A net is reversible if m_0 is a home marking, which also is satisfied by checking whether the reachability graph constitutes one strongly connected component(Tarjan's Algorithm [126]).

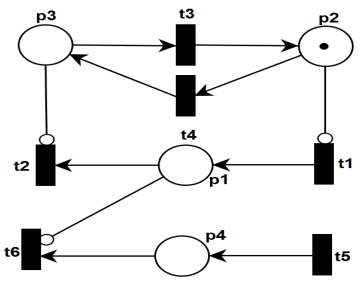
For unbounded CPTI nets, the coverability graph is not sufficient to determine the home marking and reversibility properties. This is due to the fact that the reachability and path problems can not be solved by analysing the coverability graph.

6.7.6 Transition Liveness and Quasi-Liveness

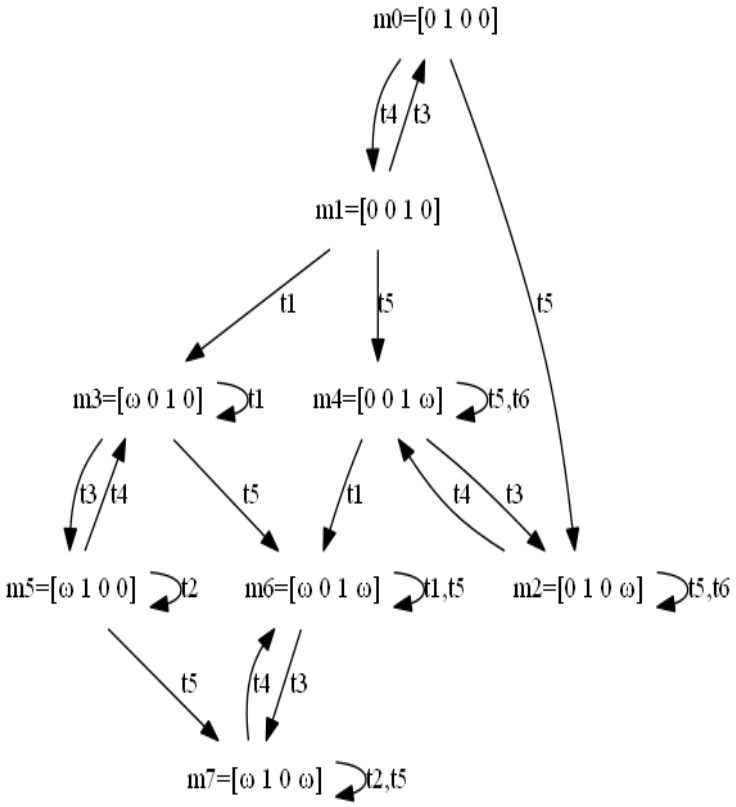
A transition is said to be quasi-live if it can be made to fire once, and it is called live if it can possibly fire from any reachable marking. These notions have already been formalised in Definition 5.10.

For a CPTI net a transition t is quasi-live if it appears in some arcs in the coverability graph. The quasi-liveness is thus decidable for CPTI nets.

The liveness analysis for P/T nets (see Section 5.3) also holds for



(a)



(b)

Figure 6.13: The problem of finding a path between two markings from $m_a = [4013]$ and $m_b = [3102]$, covered by m_6 and m_7 respectively

CPTI nets. That is, for bounded CPTI nets, transition liveness is decidable by means of the reachability graph. While for unbounded CPTI nets, the coverability graph is not sufficient for deciding transition liveness.

6.8 Mutually Inhibited Cohesive PTI Nets

In the previous section we defined the CPTI class of nets, and showed which properties can be decided on it. We found out that the marking reachability, path, home marking, reversibility and liveness are all properties that cannot be decided using the coverability graph.

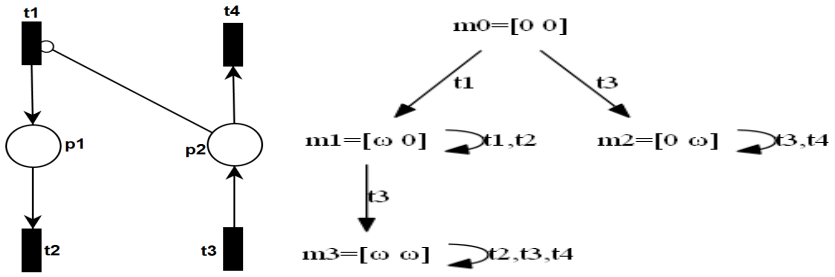
This section presents a subclass of CPTI nets, on which the above mentioned properties become decidable by coverability graph analysis. We call such a class for Mutually Inhibited Cohesive Place/Transition Nets with Inhibitors (MICPTI nets).

In CPTI nets, a place p inhibiting a transition t cannot belong to the same elementary structure (fes or ces), this also holds for MICPTI nets. However MICPTI nets have an additional restriction which states the following: A place p can only inhibit a transition t which deposits tokens, and that each inhibited transition t should deposit tokens in some place p' which in turn inhibits a transition t' that again deposits tokens in p. These transition restrictions happen in a mutual relation, which is why we have chosen to call this class Mutually Inhibited CPTI.

The main motivation behind this class of nets is that in addition to the properties of CPTI nets, the reachability and the path problems are solvable. From a fundamental point of view CPTI and MICPTI are different with respect to the monotonicity properties they satisfy. Recall that a CPTI satisfies the $T - monotonicity$ (Property 6.3), which states that from a given marking, if by firing a transition t we can reach a larger marking, then t can fire again from the generated marking.

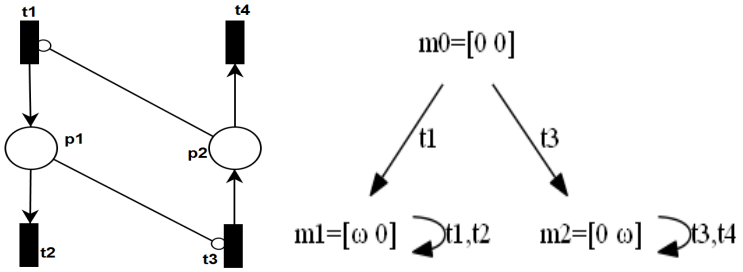
In the case of MICPTI we have an even stronger monotonicity ($S-monotonicity$), which says that from a given marking, if by firing a transition t we can reach a larger marking, then t can fire from any

larger marking which comes after the firing of t. This means that after firing t and if it is possible to fire another transition t' which leads to an even larger marking than t, t can also fire from the marking generated by t'. Figure 6.14 shows a CPTI and its corresponding coverability graph. We can notice that from m_0, t_1 can fire arbitrarily many times as reflected in m_1, however from m_1, firing t_3 leads to $m_3 > m_1$ but the firability of t_1 is lost, meaning that the $S-monotonicity$ is not satisfied.



(a) (b)

Figure 6.14: $T-monotonicity$ is satisfied but not the $S-monotonicity$



(a) (b)

Figure 6.15: The $S-monotonicity$ is satisfied

Transforming the net of Figure 6.14 to a MICPTI net can be obtained by adding an inhibitor arc from p_1 to t_3 as shown in Figure 6.15. Later

on in this section, we will show how the $S-monotonicity$ allows us to determine the path between two reachable markings.

Another effect of the restriction imposed by MICPTI is that inhibitor arcs only involve transitions that fill some place. Meaning that a transition which only removes tokens from a place without depositing tokens in another place can not be inhibited. We shall see that a consequence of this restriction allows us to solve the reachability problem.

Formally, a MICPTI net is defined in Definition 6.5 as a CPTI from Definition 6.3 with extra constraints on the use of inhibitor arcs:

Definition 6.5 (Definition of MICPTI nets)**.**
A MICPTI net is a CPTI from Definition 6.3 where:

$\forall p \in P,\ t \in p^{\circ} \Rightarrow \exists p_i \in t^{\bullet},\ \exists t_i \in p_i^{\circ},\ p \in t_i^{\bullet}$, *i.e. for a transition* t *which is inhibited by* p, t *must output to a place* p_i *which inhibits* t_i *which again outputs in* p.

Algorithm 6 (Section 6.13) determines whether a PTI net is MICPTI or not. The algorithm uses the function $isCPTI(P,T,I)$ from Algorithm 4, followed by a procedure to verify the constraints on inhibitor arcs as defined in Definition 6.5.

6.9 Monotonicity of Mutually Inhibited CPTI

Any CPTI net satisfies the $T-monotonicity$, as stated by Property 6.3. The $T-monotonicity$ is what makes it possible to apply Algorithm 5 and generate a coverability graph for unbounded CPTI nets.

MICPTI net constitute a subclass of CPTI nets which by definition satisfy the $T-monotonicity$. In addition, MICPTI nets satisfy another montonotonicity. We call it the $S-monotonicity$ which we define and discuss in this section.

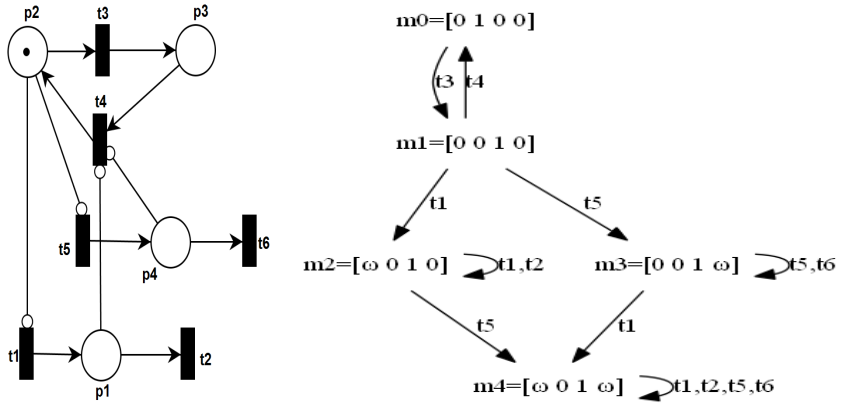
Definition 6.6 (Definition of S-monotonicity)**.**
A PTI net satisfies $S-monotonicity$ *if:*

$\forall m_1,\ m_2,\ m_1 \neq m_2,$ *if* $m_1 \rightarrow^{t_1} m_2 \wedge m_1 < m_2 \Rightarrow \forall m > m_2,$ $\exists m' > m\ s.t\ m \rightarrow^{t_1} m'$, *i.e if from* m_1 *we can fire* t_1 *to reach* $m_2 > m_1$ *then* t_1 *can fire from any marking larger than* m_2 *and reach an even higher marking.*

Note that the difference between $S-$*monotonicity* and the monotonicity of P/T nets is that for a P/T if a transition t can fire from a marking m then t can fire from any marking $m' > m$. While for a MICPTI, if a transition t can fire from m it can fire from any marking $m' > m$ under the condition that t has been fired to reach a marking $m' > m$.

Property 6.4.
Any MICPTI net satisfies the S-monotonicity.
NB! See proof 6.7 in Section 6.12.

Figure 6.16 shows the $S-$*monotonicity* of a MICPTI net. Notice that from m_1 firing t_1 leads to $m_2 > m_1$ which according to $S-$*monotonicity* means that t_1 can fire from any marking larger than m_2, which includes m_4. Likewise, with t_5 because it can fire from m_1 to $m_3 > m_1$ then we can conclude that it is also fireable from m_4.



(a) (b)

Figure 6.16: MICPTI and $S-$*monotonicity*

6.10 Analysis of Mutually Inhibited CPTI

We have shown in Section 6.7 that for CPTI nets, boundedness, deadlock freeness, and transition quasi-liveness are decidable using the coverability graph. We have also shown that marking reachability, finding a path, home marking, reversibility and liveness can not be decided using the coverability graph. However, these properties become decidable for MICPTI nets, and this section shows how they can be determined.

6.10.1 Reachability in MICPTI

The marking reachability is stated as follows: Given a marking m and a net N is m reachable (see Definition 5.8)? We have earlier mentioned that for unbounded P/T nets this problem cannot be decided by coverability graph analysis. This result also holds for CPTI nets, as discussed in Section 6.7 and formalised in Proposition 6.3. In this section we show that for a given MICPTI net, the coverability graph contains sufficient information for deciding the marking reachability.

Recall that, when we discussed the marking reachability for CPTI nets in Section 6.7, we arrived to the following result:

1. If m is explicitly present in the coverability graph, then it is a reachable marking. This can be checked by inspecting the graph nodes.
2. If m is covered by some extended marking in the coverability graph, this does not necessarily imply that m is reachable.
3. If none of the above cases hold then m is not reachable.

For MICPTI nets the seconds case is different and states that if m is covered by some extended marking in the coverability graph, then m is reachable.

Informally, the reachability problem for MICPTI is decidable due to following reasons:

– MICPTI nets uses only unweighted arcs (arc with weight one), consequently the number of tokens in a place can either be incremented or decremented by 1, and thus the symbol ω can

take any natural number.
- The fact that a transition can deposit or remove tokens from exactly one place means that the ω's inserted at different coordinates of a marking are independent. That is, any ω could be any natural number independently of the actual value of any other ω.
- By construction, a transition that only removes tokens can not be inhibited. That is, from any reachable marking and as long as the place of concern contains tokens, it is possible to fire a transition which removes tokens from that place.

Proposition 6.4 (Reachability)**.**
For a MICPTI net, every marking m that is covered by an extended marking is a reachable marking.

NB! See proof 6.8 in Section 6.12.

Consider the MICPTI net and its corresponding coverability graph in Figure 6.16. Based on Proposition 6.4 we can conclude that $m = [3\ 0\ 1\ 5]$ is a reachable marking because it is covered by m_4. While $m' = [3\ 1\ 0\ 5]$ is not because it is neither covered by any extended marking nor equal to any marking.

6.10.2 Path Problem for MICPTI

We have seen that for MICPTI nets, it is possible to determine whether a marking is reachable or not. This means, that for a given MICPTI net we can determine whether there exist a sequence of transition firing that leads to a given marking. We will now show how to obtain such a sequence of firing.

In general, the path problem for Petri nets as introduced in Section 5.3 consists of the following: Given two markings m_a and m_b, find a sequence of transition firing from m_a to m_b. When the Petri net is bounded, and its dynamic is captured in a reachability graph, the sequence of firings can be reduced to a path in the graph starting from m_a to m_b.

We have also seen that for unbounded P/T nets, and for CPTI nets the coverability graph does not embed sufficient information to determine either the marking reachability nor the path between

reachable markings. In Section 6.7.4 we presented an example showing that finding a firing sequence between m_a and m_b cannot be reduced to finding a path between two markings covering m_a and m_b respectively.

The reason for that is that the coverability graph hides information about the connection between markings, because the insertion of ω causes edges with self loops (depart and arrive to the same marking), and that these self loops hide the effects of transition firings. In the case of MICPTI nets, and due to the extra constraints on the use of inhibitor arcs, the effects of transition firings is more predictable, and thus self loop edges could be exploited to encode more information.

Intuitively, we approach the path problem by generating a so-called characteristic graph. This graph allows us to determine the edges and nodes on the coverability graph that need to be visited. We call such a path a characteristic path.

A characteristic path can be seen as a clue on which transitions are involved, but does not tell how many time each transition has to fire. This will be determined based on a simple difference between the source and the destination marking. We start by presenting the characteristic graph, and later on we show how to determine a complete path.

Characteristic Graph

Figure 6.17 shows a MICPTI net, and its corresponding coverability graph in Figure 6.18. Consider two markings $m_a = [0\ 0\ 1\ 2\ 4]$ and $m_b = [0\ 0\ 1\ 0\ 4]$. Using Proposition 6.4, we can determine that m_a and m_b are reachable, since they are both covered by a marking in the coverability graph, namely: m_7 and m_5 respectively. Further more, moving from m_a to m_b can be obtained by firing t_6 twice. However there is no edge from m_7 to m_5 on the coverability graph, and thus the fact that firing t_6 a number of times eventually leads to another marking in the coverability graph is hidden. What hides this information is that t_6 is encoded as a self-loop on m_7, despite the fact that its successive firing leads to m_5.

The characteristic graph takes advantage of the above mentioned fact, by removing all self-loops that could lead to other markings. The

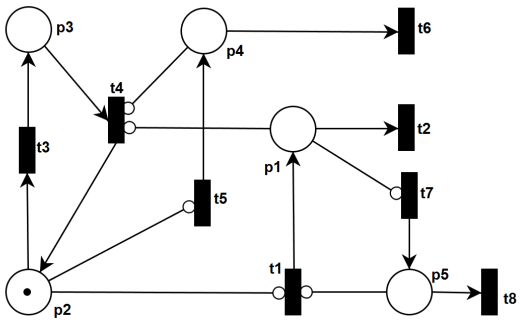


Figure 6.17: MICPTI net and the path problem

nature of MICPTI nets makes this possible, because we can a priori know which transition empties or fills which place. For example, using the net of Figure 6.17, we can know that the triple (p_4, t_5, t_6) is a fes. We can also know that t_5 adds tokens in p_4, while t_6 removes tokens from it. Thus, for a marking m, where $m(p_4) = k$, and in which t_6 can fire, firing t_6 k times leads to a marking m', where $m'(p_4) = 0$.

In practice this means that we can draw an edge between m_7 and m_5 that is labelled t_6, and read: Firing t_6 from m_7 eventually leads to m_5. Applying this idea to all the self loop edges of the coverability graph from Figure 6.18 gives the characteristic graph of Figure 6.19. This procedure is formalized in Algorithm 7 (Section 6.13).

Algorithm 7 starts by assigning the coverablility graph to the characteristic graph. Only those edges that are self loops are considered. Because these self loops only involve extended markings, we have to distinguish between those transitions that deposit tokens, and those that remove tokens. A new edge is then created between the marking of concern and a certain marking in which the place p is set to 0.

Finding a path

The characteristic graph provides a tool to determine which transition could be fired in order to move from one marking to another. We shall now exploit this graph to determine a path between two markings.

Consider again the two markings $m_a = [0\ 0\ 1\ 2\ 4]$ and $m_b =$

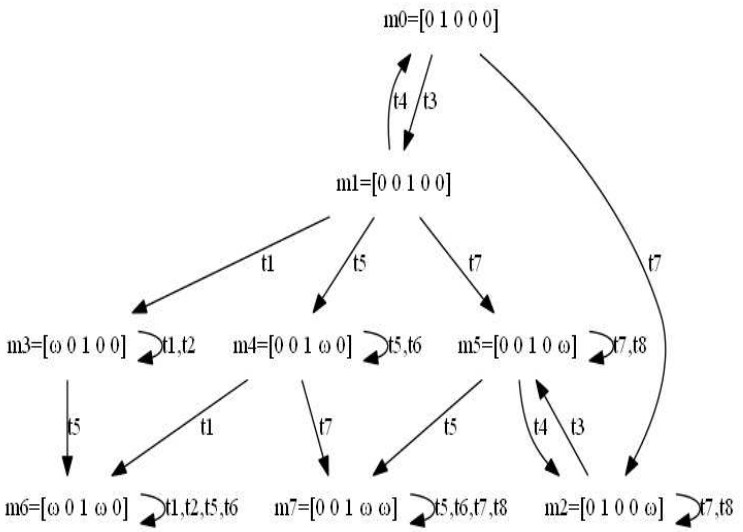


Figure 6.18: Coverability graph of MICPTI net from Fig 6.17

[0 0 1 0 4]. These are respectively covered by m_7 and m_5. Based on the characteristic graph we can find out that moving from m_7 to m_5, requires the firing of t_6, but we don't know how many times t_6 needs to fire. The number of times t_6 has to fire can be determined by computing the vector subtraction $m_b - m_a = ms_{b-a} = [0\,0\,0\,(-2)\,0]$. The fact that $ms_{b-a}(p_4) = -2$ ((-2) at the fourth coordinate) tells us that two tokens have to be removed from p_4. Since it is t_6 which removes tokens for p_4, we conclude that t_6 has to fire twice.

Of course the above mentioned example illustrate a very basic case. We will now propose a four steps procedure for determining a path between two markings. We start by a more illustrative example, and later on formalise the four steps in an algorithm.

Consider that we want to find a path between two markings $m_a = [0\,0\,1\,2\,4]$ and $m_c = [0\,1\,0\,0\,7]$.

Step 1: Subtraction

Subtracting m_a from m_c gives $m_c - m_a = ms_{c-a} = [0\,1\,(-1)\,(-2)\,3]$.

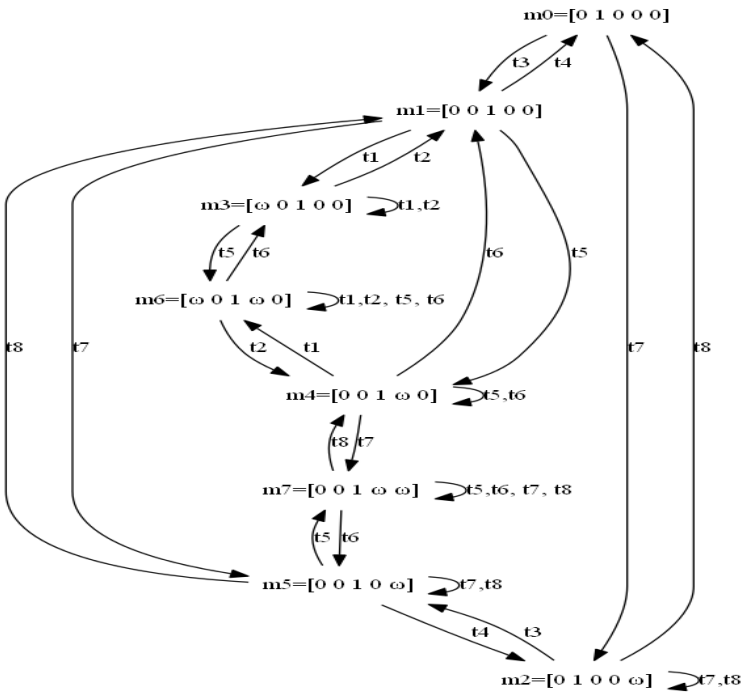


Figure 6.19: Characteristic graph of MICPTI net from Fig 6.17

Step 2: Firing Plan

$m_c - m_a$ is interpreted as follows: Having $ms_{c-a}(p_4) = -2$ and $ms_{c-a}(p_5) = 3$ means that t_6 and t_7 have to fire twice and three times respectively. However, $ms_{c-a}(p_2) = 1$ and $ms_{c-a}(p_3) = -1$ are both caused by firing t_4, but it does not mean that t_4 has to fire twice. This is because t_6 and t_7 belong to fes, while t_4 belongs to a *ces*. In a *ces* firing a transition has a double effect; it removes one token from a place and adds one token into another. There is thus no need to fire t_4 twice. As a rule we state that for any two occurrences of a transition t that belongs to a *ces* one occurrence is removed.

From the subtraction $m_c - m_a$, we obtain what we shall call a firing plan (fp). For example, we write $fp_{a \triangleright c}$ for the firing plan between m_a and m_c, where $fp_{a \triangleright c} = (t_4, t_6^2, t_7^3)$, and read: Within the path from m_a to m_c, t_4 has to fire once, t_6 has to fire twice and t_7 has to fire three times.

Step 3: Characteristic Path

The next step is to find a characteristic path from m_a to m_c, denoted as $cpath(m_a, m_c)$. Since the marking m_a is covered by m_7, and the marking m_c is covered by m_2, we find a path from m_7 to m_2 and denote it as $path(m_7, m_2) = \langle t_6, t_4 \rangle$.

Note that t_7 appears in $fp_{a \triangleright c}$, but does not appear in $path(m_7, m_2)$. This is because t_7 is represented as a self loop edge.

To obtain $cpath(m_a, m_c)$, all the transitions that are present in $fp_{a \triangleright c}$, but not present in $path(m_7, m_2)$ must be included, which is the case for t_7. Adding t_7 to $path(m_7, m_2)$ can be done by travelling the path and checking whether there exist a marking with a self loop edge labelled with t_7. When such a marking is met, t_7 must be inserted in the path. We have thus, $cpath(m_a, m_c) = \langle t_7, t_6, t_4 \rangle$, because t_7 is already present at m_7.

Step 4: The final Path

To finalise the path, each transition in the path is repeated for the number of times assigned in the firing plan. The path between m_a and m_c is then denoted as $path(m_a, m_c) = \langle t_7^3, t_6^2, t_4 \rangle$.

The Path Algorithm

The four steps for determining a path from m_a to m_b are formalised in Algorithm 8 (Section 6.13). For the sake of convenience, the algorithm has been divided in Algorithm 9 and Algorithm 10.

Correctness Analysis

Does Algorithm 8 compute a correct path for any problem instance? Intuitively we wish to claim that the answer is yes, but unfortunately, we fail to provide a formal proof. However an experimental approach is possible.

More precisely, checking whether $path(m_a, m_b)$ is a correct path, can be verified by setting the MICPTI net at m_a, execute the sequence of transitions in $path(m_a, m_b)$ and verify if that sequence actually leads to m_b. We take advantage of this fact, and try to demonstrate the correctness of Algorithm 8 by simulation.

We randomly generate 200 different MICPTI nets, using 5 numbers of fes and ces and 14 numbers of inhibitor arcs. For each net we generate 1000 problem instances of finding a path from a source to a destination marking. Each computed path by Algorithm 8 is then executed on the MICPTI net in question and checked whether it is correct. Note that the chosen size of nets is constrained by the state explosion problem. However, the size of the simulated nets remains representative to the size of nets that we target.

Moreover, we distinguish between those nets that generate one strongly connected characteristic graph and those which do not. We also keep a count on the number of times a characteristic path is found.

The results are summarized in Table 6.1. We found out that each time Algorithm 8 finds a path its was actually correct. Further more each time a characteristic path was found a path was also found. In addition we found a correlation between the strong connectivity of the characteristic graphs and paths. When a characteristic graph consists one strongly connected component, all the generated path problems had a solution and the solutions were correct indeed. When a characteristic graph consists of more than one strongly connected

Table 6.1: MICPTI path experimental results

100 strongly connected characteristic graphs

Total sim	Char path found	Path found	Path not found
100000	100000	100000	0

100 non strongly connected characteristic graphs

Total sim	Char path found	Path found	Path not found
100000	82312	82312	17688

component, not all the generated path problems had a solution. Furthermore, each time a solution was not found, it was because the source and target markings were from two distinct strongly connected components.

The results of Table 6.1 are clearly coherent. Strong connectivity of the characteristic graphs implies that each marking is reachable from any other. Even though a characteristic graph could represent infinitely many markings, it strong connectivity seems sufficient for claiming that finding a path between any two reachable markings can be determined.

This hypothesis is further strengthened, when a characteristic graph has more than one strongly connected component. Not found paths are explained by the fact that there simply does not exist a path between m_a and m_b. The problem here is that we have no way to check if path actually exists when the algorithm does not find a path.

To conclude, we wish to claim that there are good reasons for believing that the path problem is decidable for MICPTI nets, and Algorithm 8 provides an answer (Conjecture 6.1).

Conjecture 6.1 (Path)**.**
For a MICPTI net and its corresponding characteristic graph. Finding a path between two markings can be determined by Algorithm 8.

6.10.3 Home Marking and Reversibility

As a remainder from Section 5.3 and Definition 5.9, determining whether a marking can be reached from any other marking is called a home marking, and when the initial marking is a home marking, then the net becomes a reversible net. The home marking and reversibility properties are not decidable by means of coverability analysis for CPTI nets, because their decidability depends on the decidability of the reachability and the path problems.

However, based on the decidability of the marking reachability from Proposition 6.4, and the path problem from Conjecture 6.1, the home marking and reversibility become decidable for MICPTI nets. Furthermore, these problems can be reduced to the analysis of the characteristic graph.

For a MICPTI net, a marking m is a home marking if it is reachable from any marking on the characteristic graph. If the initial marking m_0 is a home marking then the MICTPI net is reversible. Finally, if the characteristic graph constitutes one strongly connected component, then each marking is a home marking and the net becomes reversible.

6.10.4 Transition Liveness and Quasi-Liveness

Recall that a transition is live if it can eventually fire from any reachable marking, and that this problem is not decidable by means of the coverability analysis for CPTI nets. However, it is decidable for MICTPTI nets using the characteristic graph.

For a MICPTI net, a transition t is live, if from any marking on the characteristic graph there exists a path in which t appears. As stated earlier in Section 5.3, a quasi-live transition t in a reversible net is live. If all transitions of a reversible net are quasi-live, then the net is live.

6.10.5 An Example

In the introduction of this chapter we presented a PTI net capturing part of the rig dynamics (see Figure 6.1). By means of the MICPTI

nets and their analysis it becomes possible to study such a model.

The net of Figure 6.1 represents three interacting parts of the drilling system: the power-slips, the top-drive, and the draw-works. Activating the power-slips suspends the drill-string. Using the top-drive the RPM can be increased and decreased, while using the draw-works we can increase and decrease the drill-string downward speed.

First, running Algorithm 6 on the net from Figure 6.1, we can determine whether the net is an MICPTI net, which is the case indeed. Furthermore, Algorithm 7 generates the net's characteristic graph as shown by Figure 6.20. Studying the characteristic graph we can

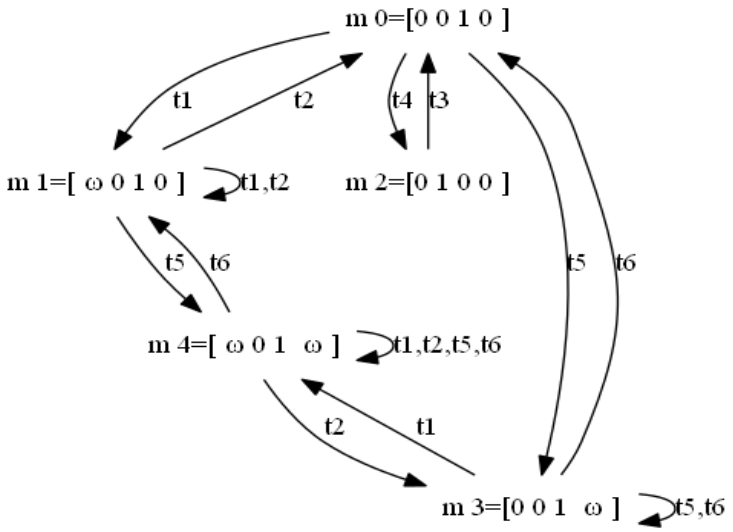


Figure 6.20: Characteristic graph of MICPTI net from Fig 6.1

determine the following general properties:

- The net is unbounded, because ω appears in some markings.
- Every transition is quasi-live, because every transition appears as an edge on the characteristic graph.
- The net is reversible, because the characteristic graph constitutes one strongly connected component.
- Since every transition is quasi-live and the net is reversible, then every transition is live and thus the net is live.

One can also specify whether some undesired states can be reached. For example, is it possible to have a state where both the power-slips and the drill-string rotation are activated? This can easily be verified by inspecting the characteristic graph for a marking in which p_1 and p_2 are set at the same time.

The point with this example is that the MICPTI class makes it possible to fully study such a model, something that was not possible before its introduction.

6.11 Chapter Summary

This chapter presented the fundamental problem of using inhibitor arcs. Their main advantage is their ability to capture system dynamics that P/T nets fail to capture. However, the main drawback of PTI nets is their Turing equivalence, which makes almost all properties of interest undecidable.

Motivated by the need of modelling specific situations that require inhibitor arcs, we moved further and explained why the coverability graph algorithm (Algorithm 2) does not work for PTI nets. We have then introduced a new class called CPTI nets. This class of nets belongs to PTI nets, and has specific restrictions on the way transitions and places are connected, and also on the way inhibitor arcs are used. We have presented a coverability graph algorithm (Algorithm 5) that apply for CPTI nets, shown why it actually works, and which model properties could be decided by it. We have found out that boundedness, quasi-liveness and deadlock freeness are decidable for CPTI nets.

The main problem with CPTI net is that marking reachability and finding path between two markings can not be solved using the coverability graph, and thus liveness, home marking and reversibility can not be decided neither. This limitations constitute a motivation to introduce MICPTI nets which are subclass of CPTI nets with a stronger restriction on the use of inhibitor arcs. As a result all problems of interest become decidable.

6.12 Proofs

6.12.1 CPTI Proofs

proof 6.1. *of Property 6.1*
The proof follows from the definition of CPTI. A transition t belongs to one and only one structure, and the allowed structures are either ces or fes. In a fes, a transition either consumes or deposits tokens (but not both) in one place, which obviously answers Case 1 and Case 2. In ces a transition consumes and deposits one token from two distinct places, which obviously leads to a marking which is not comparable to its predecessor.

proof 6.2. *of Property 6.2*
Proof by counter example. Consider two flat structures fes_1 and fes_2. The structure fes_1 has transition t_1 which fills a place p_1, while fes_2 has t_2 which fills p_2. Consider a marking m from which it is possible to fire t_1 and t_2, by firing t_2 we reach another marking $m' > m$. If p_2 has an inhibitor arc to t_1, which is possible because t_1 and p_2 belong to two different structures. Having a situation where t_1 can not fire any more, means that we have reached a marking $m' > m$ and that t_1 can not fire from m', a violation of the strict monotonicity property. We conclude that the strict monotonicity does not hold for CPTI.

proof 6.3. *of Property 6.3*
We suppose that the T-monotonicity does not hold, that is, $m_1 \rightarrow^t m_2$ with $m_1 < m_2$ and that $\nexists m_3$ such that $m_2 \rightarrow^t m_3$ with $m_3 > m_2$ and derive a contradiction. In words, we have that from a marking m_1 it is possible to reach $m_2 > m_1$ by firing t, but we can not reach a marking $m_3 > m_2$ by firing t again.

Based on CPTI definition, and Property 6.1 we have two structures fes and ces. In a ces structure it is not possible to have $m_1 \rightarrow^t m_2$ and $m_1 < m_2$, because in a ces each transition consumes one token from one place and deposits one token in another place, which means that a marking and its immediate successor are not comparable (see Case 3 in Definition 6.1). In a fes structure it is possible to have $m_1 \rightarrow^t m_2$ with $m_1 < m_2$, and since t deposits one token in one unique place p, and consumes no token from other places

we have then $\forall p_i \neq p \in P$*:* $m_1(p_i) = m_2(p_i)$ *and* $m_1(p) < m_2(p)$*. If* t *can not fire from* m_2*, it means that* m_2 *has a token in a place that inhibits* t *from firing, and that place must be* p *as it is the only one involved. But by the definition of CPTI it is not possible to have an inhibitor arc within the same structure, thus we reach a contradiction.*

proof 6.4. *of Proposition 6.1*
By contradiction, we suppose that the computed graph of a CPTI is infinite and derive a contradiction. Let G *be an infinite graph, computed by Algorithm 5. Because the number of transitions is limited,* G *has necessary a finite degree.*
According to Lemma 6.1, since G *is infinite with a finite degree, it must contain an infinitely long path with no repeated vertices.*
Using Lemma 6.2 we have that every infinitely long path with no repeated vertices contains an infinitely long and increasing subsequence with respect to the ordering relation $\leq$ *between markings.*
Let σ *be such an infinitely increasing sequence in* G*, and let* m_i *be any marking in* σ *and* m_j *its successor. We have then that* $m_i \leq m_j$*. By construction* $m_i \neq m_j$*, because Algorithm 5 does not process already existing nodes, and thus* $m_i < m_j$*. According to Property 6.1,* $\exists! p$ *such that* $m_i(p) \neq m_j(p)$ *and* $m_i(p) < m_j(p)$*. Further more, Algorithm 5 is such that whenever* $m_i(p) < m_j(p)$ *is encountered, the symbol* ω *is inserted. Comparing* m_i *and its successor* m_j *in* σ *can only be repeated for at most the number of places in the CPTI net. We have then that at most, all the places contain the symbol* ω *beyond which the case* $m_i < m_j$ *can not occur any more. We reach a contraction, the sequence* σ *is not an infinitely increasing sequence, and must thus be finite. By applying Proposition 6.1 the graph* G *must be finite. Since Algorithm 5 computes a finite graph, the algorithm terminates.*

proof 6.5. *of Proposition 6.2*
The proof follows from the behavioural rules of Property 6.1 and $T-$*monotonicity in Property 6.3. Let* m *be a reachable marking and* m' *its successor. The reachable marking* m' *necessarily emerges from the behavioural rules of Property 6.1. That is,* $m' > m$*,* $m' < m$*, or* m' *not comparable to* m*. The two latter cases are not accelerated by Algorithm 5. The correctness of Proposition 6.2 is clearly conditioned by the correctness of the acceleration function for the case* $m' > m$*.*

The acceleration can be incorrect in two ways:

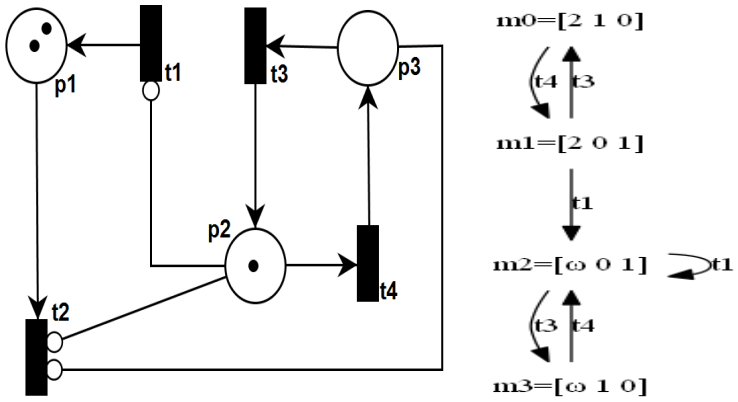
1. *ω was inserted in a place that is bounded.*
2. *Inserting ω can cause loss of information on the fireability of some transitions. That is, at some stage of the computation ω was inserted for a place p which hides the possibility that p could contain no tokens and thus hides the fireability of a transition which is inhibited by p.*

We show that none of the above cases can occur. The first case is guaranteed by the $T-$monotonicity of CPTI nets from Property 6.3 which has already been proven.

We suppose that the second case can occur and derive a contradiction:

Let G be a graph computed by Algorithm 5, and let $m \in G$ be a marking in G, and p_i a place such that $m(p_i) = \omega$. The presence of ω is by definition due to an acceleration, and the acceleration clearly concerns only the basic flat structure. Recall that in a fes structure there is only one place p_i involved, one transition t_a which only deposits tokens and one transition t_b which only removes tokens. The existence of a reachable marking m_g such that $m_g(p_i) = 0$ and that m_g is not contained in G ($m_g \notin G$), tells us that prior to the ω insertion, p_i contained at least one token, and that t_b was not enabled before ω was inserted. Based on the CPTI definition from Definition 6.3, t_b is not enabled for one reason; it is inhibited by a place $p_j \neq p_i$. So, prior to the $m(p_i) = \omega$, p_j could not be emptied, because by emptying p_j, t_b would be enabled, and by successive firings we end up with $p_i = 0$, contradicting our assumption. Following the same reasoning, not being able to empty p_j means that it is inhibited by some place p_k that can not be emptied. Because emptying p_k would enable the transition that empties p_j, and thus enable t_b leading to $p_i = 0$, which again contradicts our assumption. So, we must have $p_k \neq 0$, $p_j \neq 0$ and $p_i \neq 0$, which means that it is simply not possible to reach a marking where $p_i = 0$, which contradicts our assumption. Thus, for a CPTI net, the ω insertion done by Algorithm 5 does not hide fireability of a transition, which completes the correctness of G, and thus G is a coverability graph.

proof 6.6. *By counter example (see Figure 6.21)*



(a) (b)

Figure 6.21: Marking $m = [1\ 0\ 1]$ is covered but not reachable

6.12.2 MICPTI Proofs

proof 6.7. *of Property 6.4*
We suppose that we have $m_1 \rightarrow^{t_1} m_2$ *and* $m_1 < m_2$*, and that* $\exists m_3 > m_2$ *and* $\nexists m_3 \rightarrow^{t_1} m_4$ *i.e. from* m_1 *we can fire* t_1 *to reach* m_2 *and that there is a* $m_3 > m_2$ *from which we can not fire* t_1 *and reach some larger marking* m_4*. We derive a contradiction. Obviously, we have* $m_1 < m_3$ *because* $m_1 < m_2$ *and* $m_2 < m_3$*. Also obvious is the fact that* t_1 *is part of a* fes *structure with the effect of putting tokens in exactly one place, let* p *denote this place. Since* $m_3 > m_1$ *we have* $m_1(p) \leq m_3(p)$*. Being able to fire* t_1 *from* m_1 *and not from* m_3 *means that* m_3 *marks at least one place which inhibits* t_1 *from firing, and that this place is obviously filled by some transition* t_2*, so* t_2 *has deposited a token in a place which inhibits* t_1*. But according to MICPTI definition,* t_1 *should have deposited a token in a place which inhibits* t_2*, meaning that* t_2 *could not fire from a marking where* p *contains tokens, reaching a contradiction.*

proof 6.8. *of Proposition 6.4*
By contradiction. We suppose that there exists a marking m *which is covered by an extended marking, but is not reachable, and derive a contradiction. The marking* m *is clearly not equal to any explicit*

marking in the Coverability Graph (CG), because it would turn it to a reachable marking which contradicts our assumption. Let M_ω *be the set of all extended markings, i.e. the markings containing at least one* ω *symbol in their coordinates. For example, in Figure 6.16 we have that* $M_\omega = \{m_2, m_3, m_4\}$*. The fact that* m *is covered implies that there exits a marking* $m_c \in M_\omega$ *such that* $m < m_c$*. If* m *is not reachable it means that at some place* p*,* $m(p) = k$*, where* k *is a natural number and* k *is a value that* ω *does not take. Let* $\Omega(p)$ *be the set of all the possible values that* p *can have at the extended marking* m_c*.*

Now, if m *is not reachable it means that* $k \notin \Omega(p)$*. By construction we have that a transition* t *which can cause an* ω *insertion deposits at most one token at the time. That is, the interval between elements of* $\Omega(p)$ *must be one. So again if* m *is not reachable it implies that* $\forall a \in \Omega(p)$*,* $a \in (k, \infty)$*, because otherwise* $m(p)$ *would be in the interval which would make it reachable. Again by construction the presence of* ω *concerns* fes *structures only and in a* fes *structure of MICPTI net, a transition* t' *that removes tokens can not be inhibited, i.e. if* $m(p) = k > 0$*,* t' *can fire* k *times until* $m(p) = 0$ *or simply* $k = 0$*, which means that* $\forall a \in \Omega(p)$*,* a *can be any nature number, and thus* $\Omega(p)$ *contains all natural numbers, making* m *a reachable marking. Thus we have reached a contradiction.*

6.13 Algorithms

Algorithm 4 Determining whether a net is CPTI

```
1: Initialization:
2:    ES ← ∅                                   {Set of Elementary Structures}
3: function isCPTI(P, T, I)
4:    for all p ∈ P
5:       if cond_a = false
6:          return false
                          {cond_a, cond_fes and cond_ces from Definition 6.2}
7:       if cond_fes =true
8:          t ←• p
9:          t' ← p•
10:         ES ← ES ∪ {(p, t, t')}
11:         P ← P − {p'}
12:      else
13:         if cond_ces = true
14:            t ←• p
15:            t' ← p•
16:            p' ←• t
17:            ES ← ES ∪ {(p, p', t, t')}
18:            P ← P − {p, p'}
19:         else
20:            return false
                    {If t and p are in the same es and p inhibits t return false}
21:   for all (p, t) ∈ I
22:      for all es ∈ ES
23:         if t ∈ es ∧ p ∈ es
24:            return false
25:   return true
```

Algorithm 5 Finding the Coverability Graph of a CPTI net

Initialization:
  $(V, E, v_0) \leftarrow (m_0, \emptyset, m_0)$ {*Initialize graph with m_0 as its only vertex*}
function $CovGraph(V, E, v_0)$
  $S : set \leftarrow \{m_0\}$ {*S is a set filled with m_0 at start*}
  while $S \neq \emptyset$
    select $m \in S$ {*Choose an entry m from S*}
    $S \leftarrow S \backslash \{m\}$ {*Remove the entry m from S*}
    for all $t \in enabled(m)$
      $m' \leftarrow fire(t, m)$ { *enable and fire from Def. 5.2*}
      $m' \leftarrow Accelerate(m, t, m', V, E)$
      if $m' \notin V$
        $V \leftarrow V \cup \{m'\}$ {*Add m' to the set of vertices*}
        $S \leftarrow S \cup \{m'\}$ {*Add the next marking to S *}
      $E \leftarrow E \cup \{(m, t, m')\}$ {*Connect m and m' with an arc t *}
  return (V, E, v_0) {*Return the Coverability graph*}
function $Accelerate(m, t, m', V, E)$
  if $m < m'$
    $m' \leftarrow m' + (m' - m) \times \omega$
  { *If the successor marking m' is strictly larger than its previous marking m, then all the places in m' which are larger than in m will contain ω *}
  return m'

Algorithm 6 Determining if a net is a MICPTI net

Initialization:
  $N \leftarrow PTI(P, T, I)$
function $isMICPTI(P, T, I)$
  $cpti \leftarrow isCPTI(P, T, I)$ {*From Algorithm 4*}
  if $cpti =$**false**
    return false
  for all $(p, t) \in I$
    $cond \leftarrow \exists p_i \in t^\bullet \ \wedge \ \exists t_i \in p_i^\circ \ \wedge \ p \in t_i^\bullet$ {*From Def 6.5*}
    if $cond \neq$ **true**
      return false
  return true

Algorithm 7 Finding Characteristic graph of a MICPTI net

Initialization:
  (P, T, I) {*MICPTI net* }
  (V, E) {*The coverability graph* }
function $CharGraph(V, E, P, T, I)$
  for all $e \in E$
    if $e = (m, t, m) \ \wedge \ |^\bullet t| = 1$
      $p \leftarrow^\bullet t$
      $m' \leftarrow m$
      $m'(p) \leftarrow 0$ {*Replace ω by 0 for the place p*}
      $E \leftarrow E \cup (m, t, m')$ {*Connect m and m' with t* }
  return (V, E) {*Return the characteristic graph*}

Algorithm 8 Finding a Path between two markings of a MICPTI net

Initialization:
  (P, T, I) {*MICPTI net* }
  FES {*Set of flat elementary structures* }
  CES {*Set of circular elementary structures* }
  (V, E) {*Characteristic graph* }
function $path(m_a, m_b, V, E, P, T, FES, CES)$
  $fp_{a \triangleright b} \leftarrow \emptyset$ {*Firing plan*}
  $ms_{b-a} \leftarrow m_b - m_a$ {*Step 1 subtraction*}
  $firingPlan(P, T, ms_{b-a}, fp_{a \triangleright b})$ {*Step 2 firing plan from Alg 9*}
  $cpath \leftarrow cpath(m_a, m_b, V, E, fp_{a \triangleright b})$ {*Step 3 characteristic path from Alg 10*}
  if $cpath = \perp$
    return $\perp$
  $path \leftarrow finalPath(cpath, fp_{a \triangleright b})$ {*Step 4 final path*}
  return $path$
function $finalPath(cpath, fp_{a \triangleright b})$
  $path \leftarrow cpath$
  for $t^n \in fp_{a \triangleright b}$
    find $t \in path$ and replace it with t^n.
  return $result$

Algorithm 9 Find Firing plan

1: **function** $firingPlan(P, T, ms_{b-a}, fp_{a \triangleright b})$
2: $A \leftarrow \emptyset$ {*A temporary set of transitions*}
3: **for all** $p \in P$
4: $s \leftarrow ms_{b-a}(p)$
5: **if** $p \in FES$
6: **if** $s > 0$
7: $fp_{a \triangleright b} \leftarrow fp_{a \triangleright b} \cup ({}^\bullet p)^s$ {*Add s times the transition filling p*}
8: **if** $s < 0$
9: $fp_{a \triangleright b} \leftarrow fp_{a \triangleright b} \cup (p^\bullet)^{|s|}$ {*Add s times the transition emptying p*}
10: **if** $p \in CES$
11: **if** $s < 0$
12: **if** $p^\bullet \notin A$
13: $A \leftarrow A \cup p^\bullet$
14: $fp_{a \triangleright b} \leftarrow fp_{a \triangleright b} \cup (p^\bullet)^{|s|}$ {*Add the transition emptying p to A and to the firing plan* $fp_{a \triangleright b}$}
15: **else**
16: **if** $p^\bullet \in A$
17: $A \leftarrow A \backslash \{p^\bullet\}$ {*Remove the transition emptying p from A, because it has already been considered*}
18: **if** $s > 0$
19: **if** $p^\bullet \notin A$
20: $A \leftarrow A \cup {}^\bullet p$
21: $fp_{a \triangleright b} \leftarrow fp_{a \triangleright b} \cup ({}^\bullet p)^s$ {*Add the transition filling p to A and to the firing plan* $fp_{a \triangleright b}$}
22: **else**
23: **if** ${}^\bullet p \in A$
24: $A \leftarrow A \backslash \{{}^\bullet p\}$ {*Remove the transition filling p from A, because it has already been considered*}

Algorithm 10 Find Characteristic path

1: **function** $cpath(m_a, m_b, V, E, fp_{a \triangleright b})$
2: $cpath \leftarrow \perp$ {*Will contain a sequence transition*}
3: $m_{ca} \leftarrow Cover(m_a)$
4: $m_{cb} \leftarrow Cover(m_b)$
5: $m_{ca}[\sigma\rangle m_{cb} \leftarrow \emptyset$ {*denotes sequence* $\langle m_i, t_j \rangle$}
6: **if** $m_{ca} = m_{cb}$
7: $m_{ca}[\sigma\rangle m_{cb} \leftarrow \{m_{ca}\}$
8: $\forall t \in fp_{a \triangleright b}$
9: **if** $(t, m_{ca}) \notin m_{ca}[\sigma\rangle m_{cb}$
10: Insert (t, m_{ca}) in $m_{ca}[\sigma\rangle m_{cb}$
11: **else**
12: $m_{ca}[\sigma\rangle m_{cb} \leftarrow BFS(m_{ca}, m_{cb})$ {*Shortest path using Breath-First-Search*}
13: **if** $m_{ca}[\sigma\rangle m_{cb} \neq \perp$
14: **if** $\exists t \in fp_{a \triangleright b}$ *such that* $t \notin m_{ca}[\sigma\rangle m_{cb}$
15: find m in $m_{ca}[\sigma\rangle m_{cb}$ with self loop edge (m, t, m)
16: **if** $\nexists m \in m_{ca}[\sigma\rangle m_{cb}$
17: **return** $\perp$
18: **else**
19: Insert (t, m) after m in $m_{ca}[\sigma\rangle m_{cb}$
20: **for** $t \in m_{ca}[\sigma\rangle m_{cb}$
21: $cpath \leftarrow cpath \cup \{t\}$
22: **return** $cpath$

Chapter 7

Reactive Processes

A *reactive process* is an entity that observes the *environment* and reacts using the *system*. In the drilling context, the rig represents the *system*, while the well represents the *environment*. In Chapters 5 and 6 we proposed a theory for modelling the *system*. In this chapter we present a theory for modelling the interaction between the *system* and the *environment*.

7.1 Introduction

In Chapter 4, we suggested a separation of concerns between the rig and the well. However, a closed loop between the two exists, changes in the one could cause changes in the other. Closing the loop will be addressed in this chapter.

We will use the terms *system* and *environment* to emphasize that our approach is not limited to the drilling domain only, but could also apply to other fields that share the same fundamental assumptions.

Fundamentally, we address the cases where the state space of the *system* can be explicitly captured in a DES model, and where the *environment* cannot. Furthermore, we assume that the state space of the *system* is captured in a Petri net model on which the *path* and *marking reachability* problems are decidable.

Based on these assumptions, we model the influence of the *environment* on the *system* using reactive processes. A *reactive*

process is defined as a entity that senses, does some processing and reacts in order to achieve a certain *goal*. Our hypothesis is that, by combining an appropriate set of reactive processes, we are likely to maintain a satisfactory interaction between the *system* and its *environment*.

To illustrate our idea, consider a car to be the *system*, and the weather as its *environment*. A *reactive process* could then be to reduce the speed based on weather conditions. While another *reactive process* could be to activate the windscreen wiper when its rains. Put simply, a *reactive process* role is to link between what can be observed from the *environment* and what can be done using the *system*.

The *reactive process* idea is related to the so-called Subsumption architecture that was proposed by Brooks [19]. This approach when applied to robots, leads to a layer control. The layers are organized as software modules corresponding to different levels of competence and new competences are added whenever required. Typically, a robot could be build with the competence of *never hit an object*, adding a new competence *move around* should include *never hit an object*. The emerging behaviour will then be *move around and never hit an object*.

Even though a number of robots have been successfully developed using this approach, there is no strong guaranty on unexpected behaviour [12]. In particular when competing modules are involved. For example, a module implementing the competence *move around*, could conflict with another implementing *stop when too close to a wall*. When the number of modules increases their interactions become intractable, making it difficult to guarantee a correct behaviour.

In the Petri net framework, there have been attempts to model the *environment* influence on the *system*. This was mainly done in Synchronous Petri net [30] in which the *environment* behaviour is coupled with the *system* by associating sensory data to transitions. To use the robot example again, an object proximity condition could be associated with a Petri net transition *move*, in such a way that *move* can only be executed if no object is close enough to the robot. The problem with this approach is that it becomes difficult, if not impossible to verify the system properties any more. That is, if we introduce uncontrolled conditions, and associate them with the

transition firing, we loose the *decision power*, and thus the ability to determine the legal set of states. More precisely, we can not tell any more if a transition can ever fire, because we cannot always know if some sensory values can ever be reached. To be able to give such a guaranty we need precise models of both; the *system* and the *environment*. In such case, the question is better addressed in the reactive system framework [66]. However, we can not assume that the environment behaviour is fully known, and even if that were the case we can not assume that it is captured in a transition system (Petri net or such). These assumptions are unfortunately central in reactive systems, and will therefore not adopt that approach.

In our approach, the gradually added competences can be regarded as reactive processes that focus on very specific aspects. For example, a robot could have a *reactive process* that is focused on avoiding collisions. If the robot gets too close to an object, a *reactive process* must order it to stop. Another *reactive process* could compute a new trajectory and order the robot to apply some parameters such as speed and direction. By combining these processes, we can obtain the competence *move around and never hit an object*.

The added value of our approach, is that we can systematically study the added competences, whether they conflict or subsume the existing ones. In our approach, the robot is considered as a *system* which is deployed in some *environment*. Using our assumption that the *system* dynamics is captured in a Petri net model, the robot behaviour is then contained within a set of states. This means that any requested action on the robot is per definition within the robots legal states. One can thus associate a *reactive process* to a subset of states that correspond to a certain *goal*. If we can know which set of states are associated to which *reactive process* through its *goal*, we can also determine which process conflicts or subsumes another. Simple sets operations such as set-intersection and set-inclusion will let us determine the relations between those processes.

Note that in agent-based-systems [116], the level of sensing and processing is what usually define the agent, among which we find; *reflex agent*, *goal agent*, *intelligent agent*, *learning agent*, *fuzzy agent* and more. Agent based systems, mainly focus on agent's capabilities, while we focus on their reactivity on a common *system*. So, to avoid

any confusion with agent-based-systems, we will use the term *reactive process* rather then agent. In other words, how a *reactive process* takes the decision to act is beyond the scope of this thesis. What interest us is, once a *reactive process* acts, how it does it, and how to combine several *reactive processes* taking actions on a common *system*.

7.2 Basic Notions

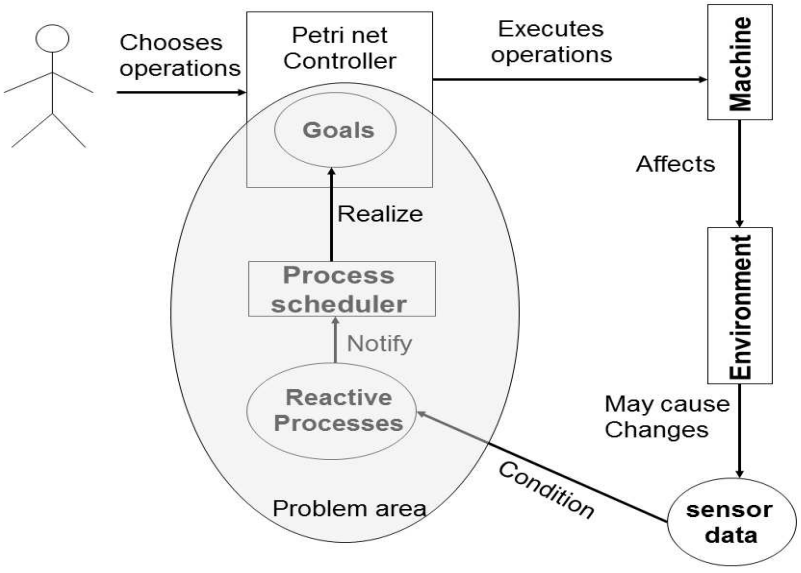


Figure 7.1: Reactive processes domain

Figure 7.1 gives an overview of our approach. As the figure shows, a reactive process takes as input sensory readings. These inputs are used by *reactive processes* to determine whether they must trigger or not. When a *reactive process* triggers, it does it by notifying the *process scheduler* which main task is to obtain the *goal* of the triggered process. A *goal* describes a set of parameters that the *controller* should apply on the *machines*. For a drilling rig, a *goal* could be to have a certain flow-rate, or a drill-string elevation of a certain speed. The *process scheduler* obtains a given *goal* by causing the *controller* to execute intermediate actions for reaching a state that corresponds

to the specified *goal*. When several *reactive processes* are involved, determining which process should obtain its *goal* before another can become complicated, and a systematic approach is needed.

To realize a *process scheduler*, we propose to first determine the relation between different *goals*, and use that information as an input for the scheduler as a constraint to respect, and also for optimizing the scheduling. For example, consider that a *reactive process* $proc_1$ has a *goal* g_1. When $proc_1$ triggers, the *process scheduler* computes a sequence of actions that the controller needs to take in order to obtain g_1. When a $proc_2$ with a *goal* g_2 triggers, it becomes important for the *process scheduler* to know whether there exists a state that satisfies both goals, and if such a state exists how it can be reached? The information needed by the *process scheduler* is the relation between g_1 and g_2.

We associate a *goal* g with a subset of the *system* states which we call Mg. The set of states that satisfy g_1 and g_2 are thus Mg_1 and Mg_2 respectively. Knowing Mg_1 and Mg_2 allows us to determine whether g_1 and g_2 can be obtained at the same time or not. Typically, having $Mg_1 \cap Mg_2 = \emptyset$ implies that g_1 and g_2 have no common states. However, having $Mg_1 \cap Mg_2 \neq \emptyset$ implies that there exist some states that satisfy both goals. In this case, the *process scheduler* could take advantage of that information and advance the *system* to some state that satisfies the two goals rather than only one.

Determining the relation between *goals* requires the decidability of the *marking reachability*. Determining the sequences of actions to reach a certain state which satisfies a certain *goal*, requires the decidability of the *path* problem. In [118, 119] we limited our selves to bounded Petri net. In this chapter we will extend the approach to include MICPTI class, since both of the *marking reachability* and *path* problems are also decidable on it.

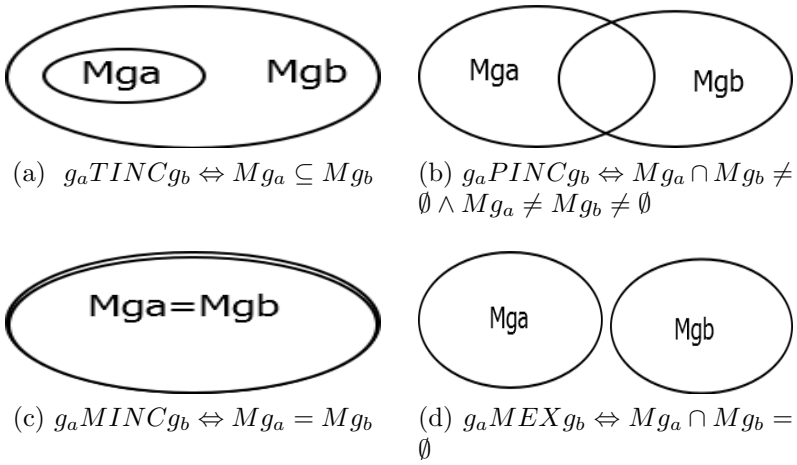
7.2.1 Goal

The input domain of a *reactive process* is defined by sensory variables, while its output domain is defined by what the *system* can do. What links a *reactive process* to a *system* is called a *goal*, which we define as a subset of the *system* states.

We denote the set of markings that satisfy a *goal* g for Mg (see Definition 7.1). If a *system* reaches a marking (state) $m \in Mg$, we say that the goal g is satisfied.

Definition 7.1 (Definition of *goal*)**.**
Let a net N be a Petri net with a set of places P and $R(m_0)$ its reachable markings. A goal g is such that $g : P_g \to \mathbb{N}$ where $P_g \subseteq P$. For $P_g = \{p_1, p_2, ...p_i\}$, we write $g = [m(p_1) = k, m(p_2) = l, ...m(p_i) = x]$, where k, l and x are number of tokens. A goal g is satisfied by a marking m, iff $\forall p \in P_g\ m(p) = g[m(p)]$. Finally, the set of markings that satisfy a goal g will be denoted Mg.

7.2.2 Relation Between Goals



(a) $g_a TINC g_b \Leftrightarrow Mg_a \subseteq Mg_b$

(b) $g_a PINC g_b \Leftrightarrow Mg_a \cap Mg_b \neq \emptyset \wedge Mg_a \neq Mg_b \neq \emptyset$

(c) $g_a MINC g_b \Leftrightarrow Mg_a = Mg_b$

(d) $g_a MEX g_b \Leftrightarrow Mg_a \cap Mg_b = \emptyset$

Figure 7.2: The Four relations between goals

Intuitively a goal g_a totally includes a goal g_b, when achieving g_a implies achieving g_b. A goal g_a partially includes the goal g_b when some of the markings that satisfy g_a also satisfy g_b. Two goals g_a and g_b are mutually inclusive when they totally include each others, or simply achieving the one implies achieving the other. Finally two goals are mutually exclusive when they can not be achieved at the same time. Figure 7.2 summarises these relations.

Definition 7.2 (Total Inclusion)**.**
Let Mg_a and Mg_b be the sets of markings that satisfy the goals g_a and g_b respectively as defined in Definition 7.1. A goal g_a totally includes g_b ($g_a TINC g_b$), if $Mg_a \subseteq Mg_b$. The function $TINC(g_a)$ defines the set of goals that g_a includes, i.e. $\forall g, g_a TINC g$, then $g \in TINC(g_a)$.

Definition 7.3 (Partial Inclusion)**.**
Let Mg_a and Mg_b be the sets of markings that satisfy the goals g_a and g_b respectively as defined in Definition 7.1. A goal g_a partially includes g_b ($g_a PINC g_b$), if $Mg_a \cap Mg_b \neq \emptyset \wedge Mg_a \neq Mg_b \neq \emptyset$. The function $PINC(g_a)$ defines the set of goals that g_a partially includes, i.e. $\forall g, g_a PINC g$, then $g \in PINC(g_a)$.

Definition 7.4 (Mutual Inclusion)**.**
Let Mg_a and Mg_b be the sets of markings that satisfy the goals g_a and g_b respectively as defined in Definition 7.1. A goal g_a mutually includes g_b ($g_a MINC g_b$), if $Mg_a = Mg_b$. The function $MINC(g_a)$ defines the set of goals that g_a has a mutually inclusion relation with, i.e. $\forall g, g_a MINC g$, then $g \in MINC(g_a)$.

Definition 7.5 (Mutual Exclusion)**.**
Let Mg_a and Mg_b be the sets of markings that satisfy the goals g_a and g_b respectively as defined in Definition 7.1. A goal g_a mutually excludes g_b ($g_a MEX g_b$), if $Mg_a \cap Mg_b = \emptyset$. The function $MEX(g_a)$ defines the set of goals that g_a has a mutually exclusive relation with, i.e. $\forall g, g_a MEX g$, then $g \in MEX(g_a)$.

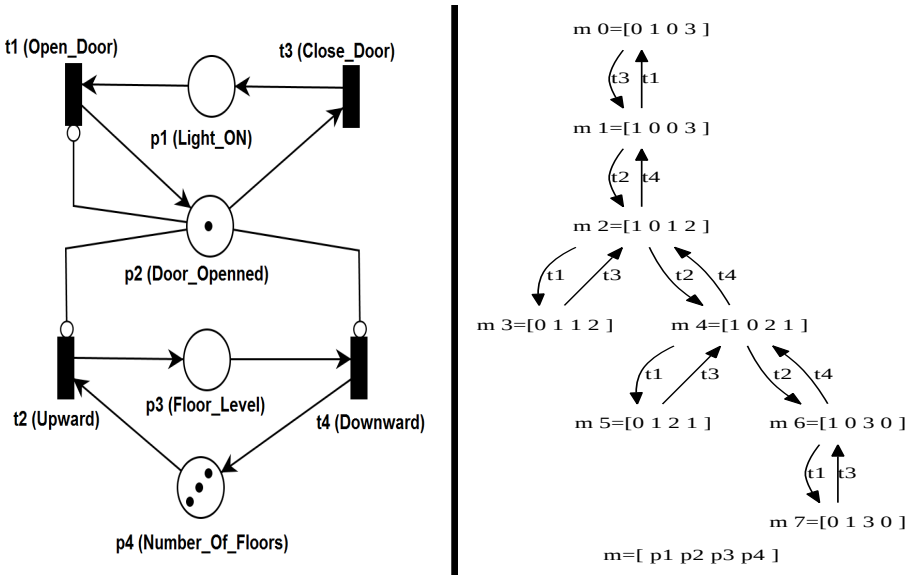
7.3 Reactive Processes and Bounded Nets

Determining the relations between goals depends on the nature of the state space: finite or infinite space. In this section we focus on finite space only, while the infinite space analysis will be treated in the next section. We assume that we have at our disposal a reachability graph, and use this graph to find out the relations between the different goals.

The existence of a reachability graph makes our approach almost trivial, because for every specified goal g we can easily determine the

set of states Mg that satisfy g. Determining Mg is no more but an application of the *sub-marking reachability* on bounded Petri nets, which was already addressed in Section 5.3.6. The relation between different goals is then determined using basic operations on finite sets such as set inclusion, equality and intersection.

7.3.1 Elevator Example



(a) (b)

Figure 7.3: A Petri net model of an elevator and its corresponding reachability graph

The Petri net model of Figure 7.3a captures the dynamic of an elevator that operates in a four floors building. The elevator can move upwards or downwards, and can have the door open or closed. Closing the door turns the light on, while opening the door turns it off. The complete set of states is contained in the reachability graph of Figure 7.3b. We define five goals:g_1, g_2, g_3, g_4 and g_5, and associate them to some concrete processes, as follows:

- $g_1 = [m(p_1) = 1]$ (light maintenance): Let g_1 be the goal of a light maintenance process. Combined with a photo sensor, the process will trigger every period of time to check whether the lamp is still working.
- $g_2 = [m(p_2) = 0]$ (elevator disabling): Combined with a motion sensor, and a fire sensor. The elevator disabling process will trigger if there is a fire in the building and no person in the elevator by keeping the door closed.
- $g_3 = [m(p_3) = 0]$ (goods delivery): When the goods delivery arrive to the building from the parking place, the elevator should be at the 0th floor.
- $g_4 = [m(p_2) = 1,\ m(p_3) = 3]$ (VIP): When a helicopter transporting a VIP lands at the roof of the building, the elevator should be at the third floor with the door open.
- $g_5 = [m(p_2) = 1]$ (air conditioning): When triggered the elevator door stays opened for a period of time.

Based on Definition 7.1 and the Petri net's reachability graph of Figure 7.3b, we can extract the informations of Table 7.1. This table shows the set of markings that satisfy each of the five goals. Using those sets, we can determine the relations between different goals. For example, the goal g_1 totally includes g_2 because $Mg_1 \subseteq Mg_2$, and since $Mg_2 \subseteq Mg_1$ as well we conclude that g_1 and g_2 are mutually inclusive. Between g_1 and g_3 there is partial inclusion relation, because $Mg_1 \cap Mg_3 = \{m_1\} \neq Mg_1 \neq Mg_3$. Finally g_1 and g_4 are mutually exclusive because $Mg_1 \cap Mg_4 = \emptyset$.

7.3.2 Short Interpretation

To understand the impact of the information proposed by Table 7.1, we need to consider the system from a designer point of view. By defining a set of *reactive processes*, a system designer can categorize the possible relations between those processes, which also allows him/her to discover system properties that were not explicitly designed. In the elevator example (Example 7.3.1), the light maintenance process has a different purpose than the elevator disabling process, but from a system state point of view, these two processes are equivalent.

Table 7.1: Relations between the Five goals

$g_1 = [m(p_1) = 1] \rightarrow Mg_1 = \{m_1, m_2, m_4, m_6\}$
$g_2 = [m(p_2) = 0] \rightarrow Mg_2 = \{m_1, m_2, m_4, m_6\}$
$g_3 = [m(p_3) = 0] \rightarrow Mg_3 = \{m_0, m_1\}$
$g_4 = [m(p_2) = 1,\ m(p_3) = 3] \rightarrow Mg_4 = \{m_7\}$
$g_5 = [m(p_2) = 1] \rightarrow Mg_5 = \{m_0, m_3, m_5, m_7\}$

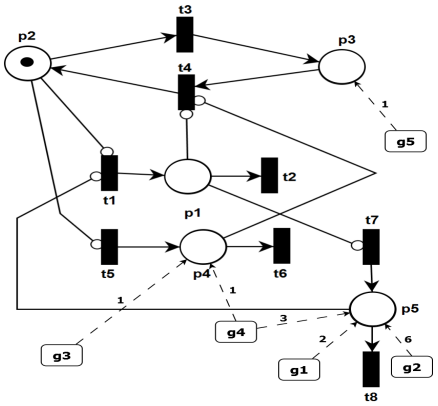
Goals	**Total Inclusion**	**Partial Inclusion**	**Mutual Inclusion**	**Mutual Exclusion**
g_1	g_2	g_3	g_2	g_4, g_5
g_2	g_1	g_3	g_1	g_4, g_5
g_3		g_1, g_2, g_5		g_4
g_4	g_5			g_1, g_2, g_3
g_5		g_4, g_3		g_1, g_2

On the other hand, conflicting processes are expressed using the mutual exclusion property (MEX). For example, the system designer could be pointed to resolve a conflict between the light maintenance, elevator disabling, and goods delivery processes.

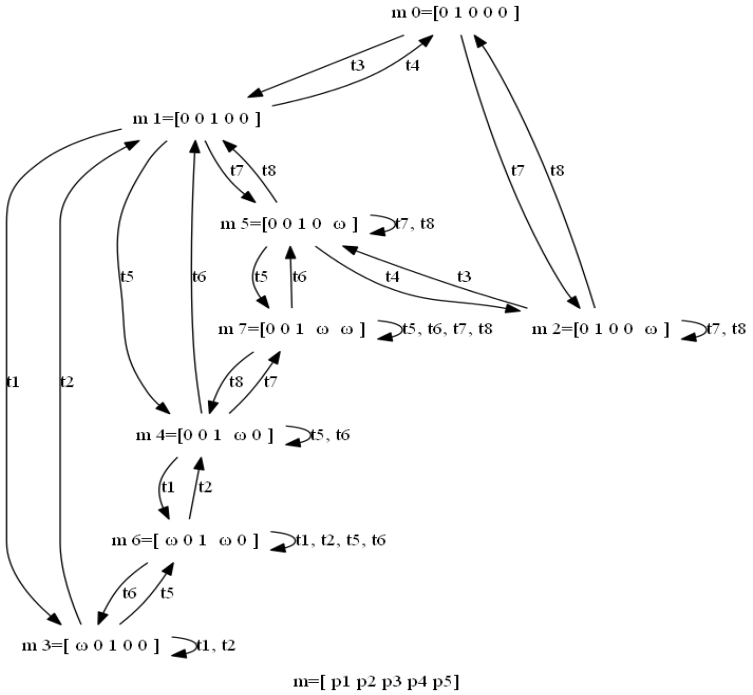
The relation between VIP and Air conditioning process describe a total inclusion $TINC$ relation, where achieving the goal of the VIP process implies achieving the goal of the Air conditioning process, but the opposite does not always hold. Finally, cases of partial inclusion can be exploited to achieve several goals such as goods delivery and air conditioning for example, where both are achieved in m_0 (door opened, 0th floor, and light off).

7.4 Reactive Processes And MICPTI Nets

In the previous section we have shown how to model *reactive processes* on top of bounded nets. In this section we show how to model them on top of MICPTI nets. MICPTI nets were introduced in Chapter 6 as possibly unbounded nets which can be fully analysed by means of their coverability and characteristic graphs.



(a) MICPTI net



(b) Characteristic graph

Figure 7.4: Goals on MICPTI nets

Representing *reactive processes* and their *goals* is done in a similar manner as with bounded nets. However in an MICPTI net, a *goal* g can have an infinite set of markings (Mg)that satisfy it. Thus, determining the relations between *reactive processes*, require sets operations (inclusion, intersection, equality) on infinite sets.

7.4.1 Determining Goals

Figure 7.4 shows a MICPTI net, its corresponding characteristic graph and five goals. Every goal has a set of markings satisfying it. Modelling a goal g on a MICPTI net is different from modelling it on a bounded net in the way the set Mg is generated. As a remainder, Definition 7.1 states that a marking m satisfies a goal g if $\forall p \in P_g, m(p) = g[m(p)]$. This can easily be verified for an explicit marking by checking the condition $\forall p \in P_g, m(p) = g[m(p)]$, which is not that obvious for extended markings. Recall that an extended marking is a marking that contains at least one ω symbol. We therefore need to determine the set of extended markings that projects on a given *goal*, and extract from them those markings that satisfy Definition 7.1. To clarify this point, we use Table 7.2 which represents the markings that satisfy the five goals from Figure 7.4.

In Table 7.2 we write $m_{i,j}$ to denote that a marking m_i projects on the goal g_j. For example, projecting the marking $m_7 = [0\,0\,1\,\omega\,\omega]$ on $g_1 = [m(p_5) = 2]$ gives $m_{7,1} = [001\omega2]$. In such a case, we say that $m_{7,1}$ satisfies g_1. This idea is formalised in Algorithm 11 (Section 7.9) which takes as input a characteristic graph CG, a goal g and outputs a set of markings Mg (extended or not) that satisfies g. The main algorithm is described in the function $Satisfy(g, CG)$. This function uses another function called $Project(g, m)$ which takes a goal and a marking as inputs, and outputs a projection m' of m. The function $Project$ returns NIL if the projection is not possible.

7.4.2 Determining Relation Between Goals

The relations between goals as defined in Section 7.2.2 depend on two set operators $\subseteq$ and $\cap$ denoting subset and intersections respectively. For two sets A and B, $A = B$ if $A \subseteq B$ and $B \subseteq A$. These

Table 7.2: Set of markings satisfying the five goals of Figure 7.4

g_1=[m(p_5)=2]: Mg_1	$m_{2,1} = [0\,1\,0\,0\,2]$ $m_{5,1} = [0\,0\,1\,0\,2]$ $m_{7,1} = [0\,0\,1\,\omega\,2]$
g_2=[m(p_5)=6]: Mg_2	$m_{2,2} = [0\,1\,0\,0\,6]$ $m_{5,2} = [0\,0\,1\,0\,6]$ $m_{7,2} = [0\,0\,1\,\omega\,6]$
g_3=[m(p_4)=1]: Mg_3	$m_{4,3} = [0\,0\,1\,1\,0]$ $m_{6,3} = [\omega\,0\,1\,1\,0]$ $m_{7,3} = [0\,0\,1\,1\,\omega]$
[m(p_4)=1, m(p_5)=3]: Mg_4	$m_{7,4} = [0\,0\,1\,1\,3]$
g_5=[m(p_3)=1] : Mg_5	$m_{1,5} = [0\,0\,1\,0\,0]$ $m_{3,5} = [\omega\,0\,1\,0\,0]$ $m_{4,5} = [0\,0\,1\,\omega\,0]$ $m_{5,5} = [0\,0\,1\,0\,\omega]$ $m_{6,5} = [\omega\,0\,1\,\omega\,0]$ $m_{7,5} = [0\,0\,1\,\omega\,\omega]$

set operations are trivial when the set of reachable markings can be enumerated. In this section we are dealing with eventually extended markings, and will therefore give special attention to how the operators $\subseteq$ and $\cap$ can be implemented.

Consider the goals g_3 and g_5 from Table 7.2, the set of markings that satisfy each of them are Mg_3 and Mg_5 respectively. In order to determine $Mg_3 \subseteq Mg_5$ we have to make sure that all the markings that are in Mg_3 are also present in Mg_5. For example the marking $m_{4,3} = [0\,0\,1\,1\,0]$ is in Mg_3, however it is also present in Mg_5, because there exists a marking in Mg_5 that covers $m_{4,3}$, for example $m_{4,5}$. The marking $m_{4,5} = [0\,0\,1\,\omega\,0]$ covers $m_{4,3}$ because for each place p $m_{4,3}(p) = m_{4,5}(p)$ or $m_{4,5}(p) = \omega$.

Definition 7.6 (Marking Containment)**.**
For a marking or an extended marking m, and a set M, we say that M contains m, if $\exists m' \in M, \forall p \in P$ s.t $m(p) = m'(p)$ or $m'(p) = \omega$.

Let Mg_a and Mg_b be two sets of markings that satisfy g_a and g_b respectively. We have $Mg_a \subseteq Mg_b$ if $\forall m \in Mg_a$, Mg_b contains m as

Table 7.3: Relations between the Five goals in MICPTI

Goals	**Total Inclusion**	**Partial Inclusion**	**Mutual Inclusion**	**Mutual Exclusion**
g_1		g_3, g_5		g_2, g_4
g_2		g_3, g_5		g_1, g_4
g_3	g_5	g_1, g_2, g_4		
g_4	g_3, g_5			g_1, g_2
g_5		g_1, g_2, g_4, g_3		

stated by Definition 7.6. Algorithm 12 (Section 7.9) takes two sets of markings A and B and determines whether $A \subseteq B$.

In order to determine the intersection between two sets of markings A and B it is not sufficient to use Definition 7.6. The reason is that in Definition 7.6 we are interested in knowing whether a marking $m \in A$ (extended marking or not) is totally contained or not in B. However two extended markings m and m' can be such that no one covers the other but still share some markings that are covered by both. For example $m_{7,1} = [0\ 0\ 1\ \omega\ 2]$ and $m_{7,3} = [0\ 0\ 1\ 1\ \omega]$, none of them totally covers the other, but $m = [0\ 0\ 1\ 1\ 2]$ is covered by both. So, to determine $A \cap B$ we have to define a new function which we call $Match(m_a, m_b)$. A marking $m = Match(m_a, m_b)$ when $\forall p \in P$, if $m_a(p) = \omega$ or $m_b(p) = \omega$ then $m(p) = Min(m_a(p), m_b(p))$ where Min stands for minimum. Otherwise $m_a(p)$ must equal $m_b(p)$. Algorithm 13 (Section 7.9) computes the intersection of the sets of markings. Note that the function $Match(m_a, m_b)$ when applied to explicit markings (without ω) computes whether $m_a = m_b$. Algorithm 14 uses Algorithm 13 to compute the intersection of a set of sets.

Going back to Figure 7.4 and Table 7.2, using Algorithms 12 and 13 we can determine the relations between goals as defined in Sections 7.2.2. The relations between our five goals are summaries in Table 7.3.

7.4.3 Feasible Path

Consider that the system state is m, and that m satisfies a goal g. If a goal g' is such that $gTINCg'$ or $gMINCg'$, it means that the marking m also satisfies g'. In such a case, the system needs not to change a state in order to obtain g'. On the other hand, if $gMEXg'$, this implies that there is no marking m' that satisfies both g and g'. However, when $gPINCg'$ it implies that there is one or more markings that can satisfy both goals, and if m is not among these markings, the system needs to advance to some state which does satisfy both goals.

We address this issue from a more general perspective, let G_m be the set of goals that m satisfies. Given a goal $g' \notin G_m$, is it possible to reach a marking m' such that $G_{m'} = G_m \cup \{g'\}$? Further more, in the path from m to m' do all markings satisfy each goal in G_m? To answer these questions, we introduce the function $feasiblepath(g_a, G_m, m, (E, V))$. This function computes a feasible path from the marking m to some marking $m' \in Mg_a$ where Mg_a are those markings that satisfy g. The computed path is such that each marking in it, satisfies each goal in G_m and that the final marking satisfies g as well.

Algorithm 15 (Section 7.9) initiates a path P to NIL and the set Mg_a to those markings that satisfy the goal g_a. The sets of markings satisfying each goal involved are collected in θ, and the set I is computed using the cumulative intersection function over θ (see Algorithm 14). For each marking $m_a \in Mg_a$ the algorithm computes a path P between the current marking m and the requested marking m_a. If the path $P \subseteq \theta$, this means that each marking in the path satisfies each goal involved. At last the path P is returned.

7.5 Scheduler Problem

Scheduling in general is a method by which processes are given access to some shared resources in order to achieve a certain quality of service [83]. Scheduling problems involve jobs that must be scheduled on machines without violating certain constraints and still obtain a quality of service.

In most cases a job is characterised by its running time and has to be scheduled for that time on one of the machines. In other cases, restrictions about job orderings are added to the picture. Efficiently scheduling a job is expressed by a so called objective function. Most often this means that the total length of the schedule is minimised, another objective function would be to minimize the waiting time or maximise the resource utilisation.

In addition, scheduling problems can also be categorised based on the process information that is available. Two types of algorithms are to be distinguished : Off-line and on-line algorithms, where the first assumes that all the arrival times and job length are known before hand, while the latter releases the job length assumption. We are concerned by on-line scheduling, as it is closer to our application domain. We therefore need a scheduler that has to react on requests with only partial knowledge about the involved processes.

What is known to the scheduler about a process is whether it has triggered or not, its goal and eventually its priority level. The set of constraints towards other processes are expressed by the relations between their goals as described in the previous sections (eg. Table 7.2 and Table 7.1), and predefined priorities for every process. We assume that the scheduler knows about those constraints, and is requested to schedule processes as they trigger. Figure 7.5 shows the basic

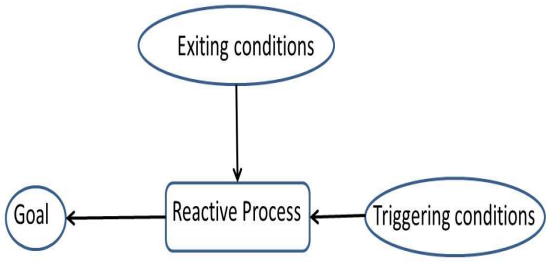


Figure 7.5: Reactive Process Elements

elements of a *Reactive Process*. This has a *goal* that is expressed in a set of states, a condition for triggering, and a condition for exiting. Such a process runs continuously, and when its triggering condition is satisfied, it requests the scheduler to satisfy its *goal*. The scheduler's

role is then to advance the system to a state that satisfies the triggered process *goal*, without violating the predefined constraints. When the exiting condition is satisfied, the *reactive process* requests the scheduler to drop the corresponding *goal*. In other word, a *reactive process* contract with the scheduler are notifications of triggering and existing, while it is up to the scheduler to realize the *goal*, without violating the relations to other process' goals and priorities.

More precisely, the scheduler maintains two queues: A *running queue*, and a *waiting queue*. When a process triggers, it is first added to the *waiting queue*, and eventually moved to the *running queue*. The scheduler has to make sure that all the processes in the *running queue* are satisfied by the current *system* state. We will show how to realize such a *process scheduler*.

7.6 Scheduling Policies

We consider two major cases; *preemptive* and *nonpreemptive* processes. We start by *nonpreemptive* processes and propose a greedy FIFO algorithm. A *nonpreemptive process* is such that once it is scheduled it cannot be suspended until its exiting condition is satisfied. In that case, a process $proc_a$ is run as it triggers, unless an already running process imposes a constraint on $proc_a$. For example, $g_b MEX g_a$, where g_a and g_b are goals of $proc_a$ and $proc_b$ respectively. If a process can not run, it is added to the waiting queue. Consequently, a process $proc_c$ that triggers after $proc_a$ and which is not in conflict with $proc_b$ can run before $proc_a$. In other words, the FIFO policy is not always respected, and is sometimes relaxed to reduce the total waiting time of processes.

For *preemtive processes* the picture is slightly different. The process $proc_c$ runs before $proc_a$ until $proc_b$ has finished, after which $proc_c$ is suspended (if in conflict with $proc_a$), and $proc_a$ is run. Priority processes work in a similar manner as with FIFO. The difference is that a process with higher priority has the advantage over a process with lower priority.

In both of FIFO and priority scheduling, whenever a process is considered, and prior to inserting it to the running or waiting sets of

processes, we have to determine if it is conflicting with the already running processes. When a running process excludes (MEX) a newly triggered process the latter is systematically added to the waiting queue. If one running process includes the coming process ($TINC$), the latter is systematically added to the running set. Otherwise ($PINC$), we have to determine a feasible path from the system state m to some state m' that also satisfies the goal of the triggered process. For this, we use Algorithm 15 which computes a feasible path based on the already running processes and the current system marking. The procedure which decides whether a waiting process w should run or not is described in Algorithm 16 and is strategy independent, i.e. both of FIFO and Priority scheduling rely on it.

Provided a set R of running processes, a temporary queue Q_{tmp} and a marking m, the process w is handled as follows:

- If $\exists proc \in R.$ s.t $w \in MEX(proc)$, then enqueue w in Q_{tmp}
- If $\exists proc \in R.$ s.t $w \in TINC(proc) \vee w \in MINC(proc)$, then add w to R
- If none of the above cases holds find a feasible path using Algorithm 15, if such a path exists add w to R, otherwise enqueue w in Q_{tmp}.

7.6.1 Basic Scenario

We define a simple scenario in order to illustrate the different scheduling algorithms. We use the model of Figure 7.3, and the goals relations of Table 7.1. The scenario shown in Table 7.4 describes five processes, their goals, triggering time and priorities respectively. In this scenario, the triggering time represents the triggering condition. For example, $proc_1$ triggers after 7 steps (or some time unit), while $proc_2$ triggers after 20 steps etc.

7.6.2 First-In-First-Out

Algorithm 17 schedules the processes following a first arrived first served policy combined with a greedy approach to minimize the average waiting time. Whether the processes involved are *preemptive* or not, plays a role and will be addressed separately in two distinct

Table 7.4: An illustrative scheduling scenario

Process	**Goal**	**Time of Trigger(steps)**	**Priority**
$proc_1$	g_1	7	0
$proc_2$	g_2	20	3
$proc_3$	g_3	13	1
$proc_4$	g_4	17	2
$proc_5$	g_5	10	1

Goals	**Total Inclusion**	**Partial Inclusion**	**Mutual Inclusion**	**Mutual Exclusion**
g_1	g_2	g_3	g_2	g_4, g_5
g_2	g_1	g_3	g_1	g_4, g_5
g_3		g_1, g_2, g_5		g_4
g_4	g_5			g_1, g_2, g_3
g_5		g_4, g_3		g_1, g_2

$g_1 = [m(p_1) = 1] \rightarrow Mg_1 = \{m_1, m_2, m_4, m_6\}$
$g_2 = [m(p_2) = 0] \rightarrow Mg_2 = \{m_1, m_2, m_4, m_6\}$
$g_3 = [m(p_3) = 0] \rightarrow Mg_3 = \{m_0, m_1\}$
$g_4 = [m(p_2) = 1, m(p_3 = 3)] \rightarrow Mg_4 = \{m_7\}$
$g_5 = [m(p_2 = 1)] \rightarrow Mg_5 = \{m_0, m_3, m_5, m_7\}$

procedures $NonPreemtiveFIFO$ and $PreemptiveFIFO$.

Non-preemptive

When a process w arrives to the waiting queue, the procedure *NonPreemtiveFIFO* attempts to schedule it using the function $Insert$, which in turn decided on whether w should run or wait. The process w will wait if there is a running process $proc_a$ that conflicts with it. If a process $proc_b$ that does not conflict with $proc_a$ arrives to the waiting queue the procedure adopts an optimistic approach an run $proc_b$ despite the fact that it arrived after w. When $proc_a$ has finished running, w will be reconsidered again, and checked for conflicts against $proc_b$. If a conflict exists w will keep waiting, because processes are *nonpreemptive*, and thus $proc_b$ can not be suspended. If no conflict exists w will run in parallel with $proc_b$.

Figure 7.6 shows how the scenario of Table 7.4 is scheduled. The vertical axis at the left is used for process identification, the right vertical axis is used to show the current system marking, while the horizontal axis describes the time steps. The process $proc_1$ is the first one that arrives and is immediately scheduled. The system marking has to move to m_1 because it satisfies g_1 while m_0 does not. When $proc_5$ arrives it has to wait, because $proc_1$ and $proc_5$ have mutually exclusive goals (see Table 7.4). When $proc_3$ arrives it is checked against $proc_1$, and since m_1 satisfies both g_1 and g_3 no need of finding a path, $proc_3$ can thus run. The same reasoning applies also to $proc_2$ when it arrives. For $proc_4$, since there exist running processes that excludes $proc_4$ ($proc_1, proc_3$ prior to triggering $proc_2$) it has to wait. In fact $proc_4$ and $proc_5$ have to wait until $proc_2$ finishes, only then $proc_5$ is chosen, and the system moves to m_0. The process $proc_5$ is chosen simply because it has arrived before $proc_4$. The last process in the waiting queue is thus $proc_4$, which can not start running before $proc_5$ has finished. The only marking that satisfies g_5 and g_4 is m_7, but as shown in the reachability graph of Figure 7.3 not all markings from the path between m_0 to m_7 satisfy g_5, and thus from m_0 $proc_4$ can not run in parallel with $proc_5$.

Preemptive

Preemptive processes require a small modification in the algorithm, as it guaranties that a waiting processes w that has triggered earlier than all running processes, must run. This is done by temporally removing the running processes that have triggered after w, and re-schedule the waiting queue. Figure 7.7 shows how the scenario of Table 7.4 is scheduled.

The first arriving process $proc_1$ is immediately scheduled, and the system moves to m_1. The next arriving process is $proc_5$ which has to wait, because g_1 mutually excludes g_5 and that $proc_1$ arrived earlier. $proc_3$ and $proc_1$ are immediately scheduled, since they do not conflict with each others nor with $proc_2$. The process $proc_4$ has to wait until $proc_1$ has finished. When $proc_1$ finishes, the process $proc_5$ is reconsidered, and causes the suspending of $proc_2$ because they are mutually exclusive and that $proc_2$ arrived after $proc_5$. The process

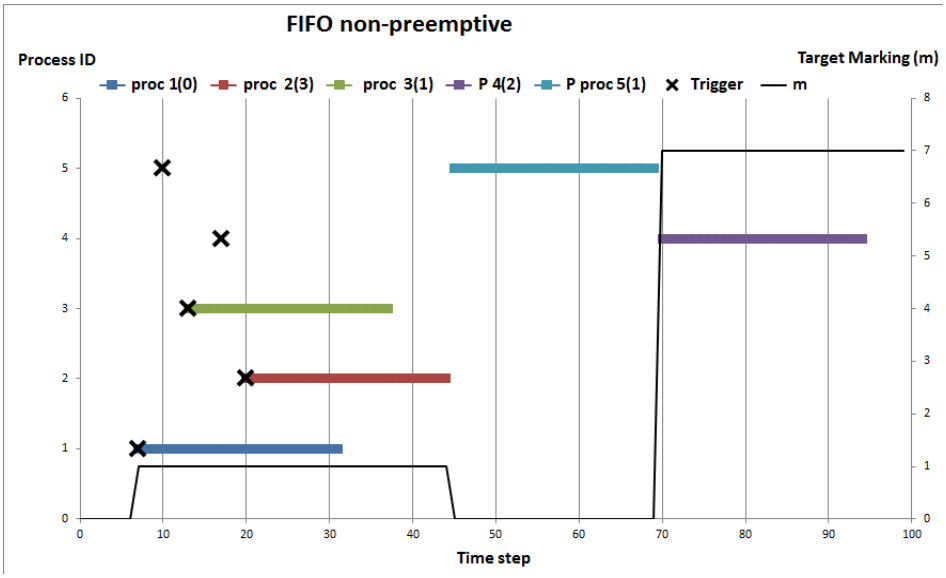


Figure 7.6: Scheduling example based on $NonPreemptiveFIFO$ procedure

$proc_3$ keeps running though because their exist a marking (m_0) and a path that satisfy both $proc_3$ and $proc_5$. $proc_4$ has to wait until $proc_5$ has finished. Resuming $proc_2$ occurs when $proc_4$ has finished.

7.6.3 Priority

Algorithm 18 schedules the processes in such a way that higher priority processes are served first. Whether the processes involved are *preemptive* or not, play a role and will be looked separately in two distinct procedures $NonPreemtivePriority$ and $PreemptivePriority$. These algorithm work almost as with FIFO, the main differences are in how the waiting queue is ordered and in the process suspending criteria which uses priority rather than triggering time.

Non-preemptive

The key idea here is that the waiting processes are ordered according to their priority. That is, when a waiting process w is considered, we

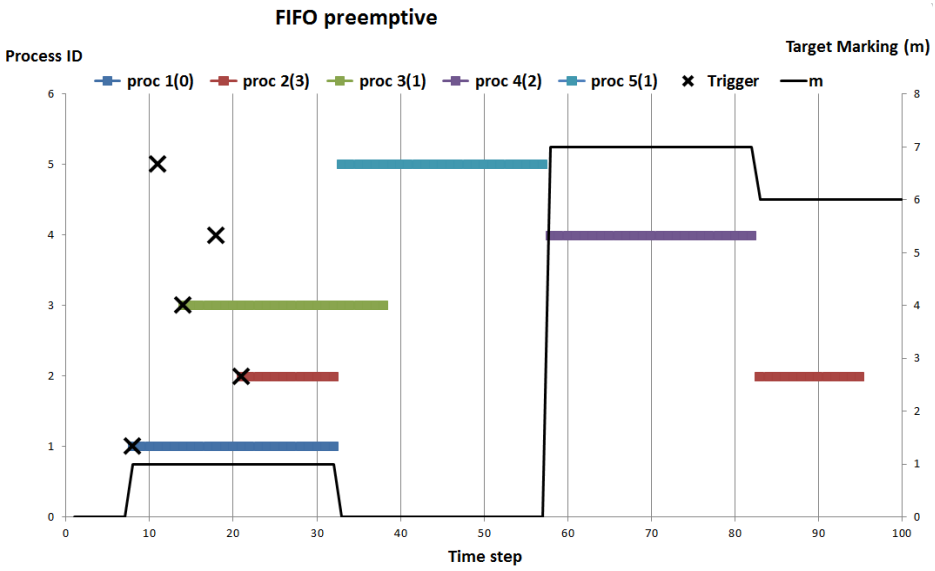


Figure 7.7: Scheduling example based on $PreemptiveFIFO$ procedure

know that it has higher priority than the next one to be dequeued. Figure 7.8 shows how the scenario of Table 7.4 is scheduled. The first arriving process $proc_1$ is immediately scheduled, and the system moves to m_1. The next arriving process is $proc_5$ which has to wait, because g_1 excludes g_5. Even if $proc_5$ had a higher priority than $proc_1$ it could not run, because processes are *non-preemptive*. Upon arrival $proc_3$ is immediately scheduled, since it does not conflict with $proc_1$. The process $proc_2$ is immediately run because it does not conflict with any running process. $proc_4$ and $proc_5$ have to wait until $proc_2$ finishes and this for three reasons, $proc_2$ has a higher priority, the mutually exclusive relation, and that the processes are non-preemptive. Even if $proc_5$ arrived before $proc_4$, the latter is considered first, because it has a higher priority. However, because $proc_4$ includes $proc_5$, the latter can also run.

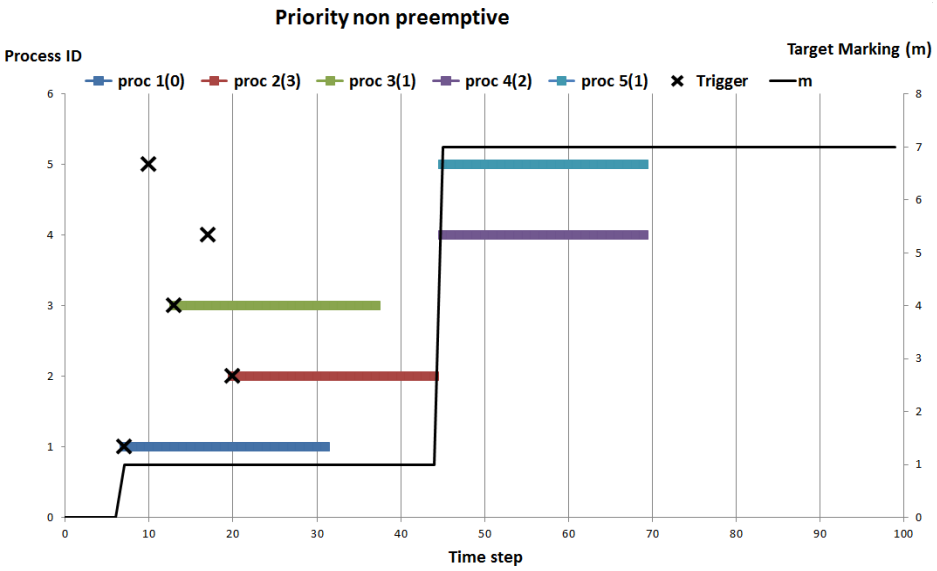


Figure 7.8: Scheduling example based on $NonPreemptivePriority$ procedure

Preemptive

The fact that a process can be interrupted guaranties that higher priority processes will immediately run. Figure 7.9 shows how the scenario of Table 7.4 is scheduled. The first arriving process $proc_1$ is immediately scheduled, and the system moves to m_1. The next arriving process $proc_5$ causes the suspending of $proc_1$ as it has a higher priority and conflicts with $proc_1$. When $proc_3$ arrives it is run in parallel with $proc_5$, until $proc_4$ triggers, and since $proc_4$ has a higher priority so far, it must run. The marking satisfying $proc_4$ is m_7 which satisfies $proc_5$ but not $proc_3$, and $proc_3$ is thus suspended. When $proc_2$ arrives it has to run immediately because it has the highest priority. Because $proc_4$ and $proc_5$ conflict with $proc_2$ they have to be suspended, and because $proc_3$ and $proc_1$ do not conflict with $proc_2$ they are resumed. When $proc_2$ finished, $proc_4$ is resumed and run in parallel with $proc_5$.

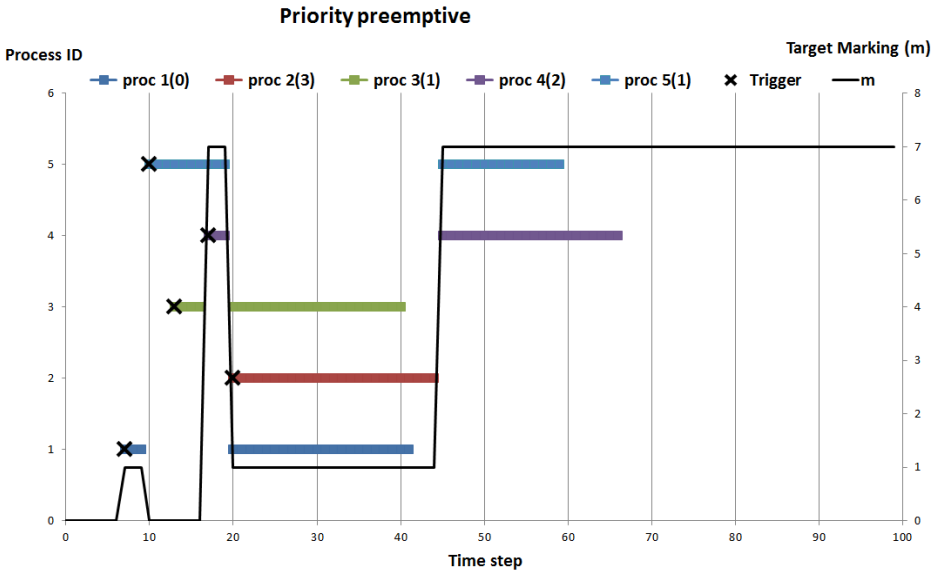


Figure 7.9: Scheduling example based on *PreemptivePriority* procedure

7.7 System Realisation

We want to show that systems can be realised using distinct reactive processes. In fact, we want to reduce the task of designing such a system to the task of composing with basic *reactive processes*. We will demonstrate how such a system can be designed using a relatively simple case, and yet rich in illustration.

In the introduction of this chapter we motivated an approach that aims at realizing a reactive system, when the environment is not fully mastered. We have further related our approach to the so-called subsumption architecture. However, in the subsumption architecture different modules of competences can have conflicting goals. In our approach, modules of competences are represented using *reactive processes*. Conflicting and none conflicting processes are formally studied by means of four relations: mutual inclusion ($MINC$), mutual exclusion (MEX), partial inclusion ($PINC$), and total inclusion ($TINC$). Determining the relations between different *reactive processes* is based on the assumption that the *system* in

question is either modelled using a bounded Petri net, or a MICPTI net. The role of those Petri net models is to capture the logic control of the *system* regardless of the *environment*, while the role of *reactive processes* is to account for the *environment* and trigger actions on the *system*.

A *reactive process* $proc_i$ is defined using four parameters: a triggering condition tc_i, an exiting conditions ec_i, a priority $prio_i$ and a goal g_i such that:

- If the triggering condition tc_i is present, then $proc_i$ tries to obtain g_i.
- If the exiting condition ec_i is present, then $proc_i$ does not need to obtain g_i.
- Higher priority processes must run before lower priority ones.

Figure 7.10 illustrates the basic components, where triggering processes are handled by a *Process Monitor* which in turn is responsible for invoking the *process scheduler*. Upon invocation the scheduler is requested to produce a set of running processes R, a set of waiting processes Q, and a path from the system current marking m to a target marking m' ($Path(m, m')$). The produced path is a sequence of commands that are executed by the *Command Executor* component. The executed commands affect the system environment and reflect in sensory data. Finally, the processes involved use sensory data to determine whether they should notify the *Process Monitor* to exit or trigger.

7.7.1 Case: Garbage Transport System

Consider a wheeled container that has a carrying capacity of 50 kg. It can fill or empty garbage at some rate. The container moves forward or backward on a horizontal axe of 30 meters by adjusting its forward and backward speeds. The set-up is equipped with weight, and position sensors (see Figure 7.11). We show how to make the garbage container repetitively move garbage from the right side to the discharging container using simple dedicated *reactive processes*.

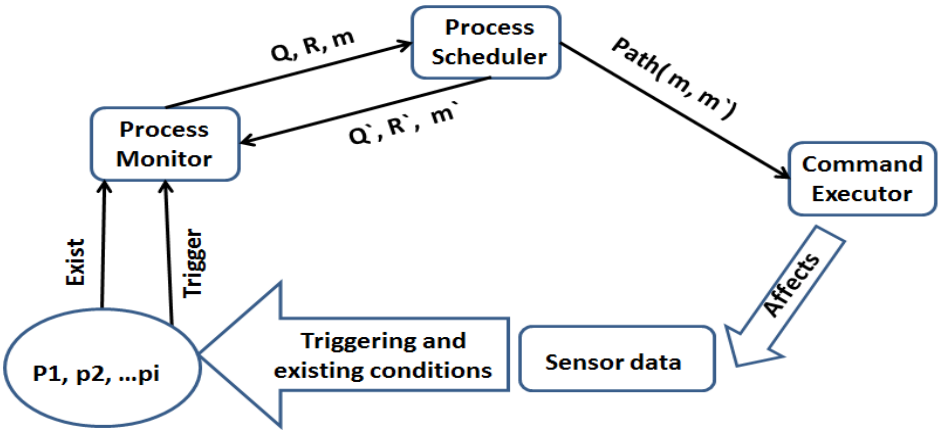


Figure 7.10: Waiting processes are set in Q and running in R, while m is the system state. The scheduler generates Q', R' and also a path from m to m'.

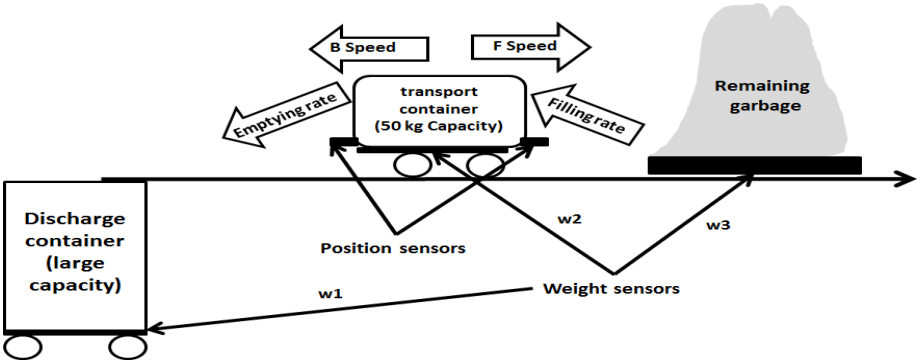


Figure 7.11: Garbage Machine Schematic

7.7.2 Petri net model

We start by capturing the dynamic behaviour of the transport container (the *system*)regardless of its *environment*. Figure 7.12a shows an MICPTI net representing the garbage transport system. The commands that can be sent to the controller are encoded in Petri net places. While Petri net transitions are used to change commands.

Moving the transport container forward is done by setting the variable F-SPEED, while moving backward is done using the variable

B-SPEED. It is not possible to set F-SPEED if B-SPEED is set and vis versa. Clearly, both speeds can not be set if the Brake is applied (BRAKE-ON, BRAKE-OFF). The filling and emptying rates, are controlled using the variables F-RATE and E-RATE respectively. Before setting any of those two variables the brake has to be on. Finally, it is not allowed to fill and empty at the same time. The commands and their units are summarised in Table 7.5. We also use four sensors, one for the position denoted *pos* and three others for weights denoted w_1, w_2 and w_3.
As the model of Figure 7.12a shows no assumption is taken about the sensor data when representing the *system*. All what is considered are the commands that the system can take. A representation of the states is shown in the characteristic graph of Figure 7.13b. We assimilate the system to an infinite system by letting the speeds and the rates take any positive value.

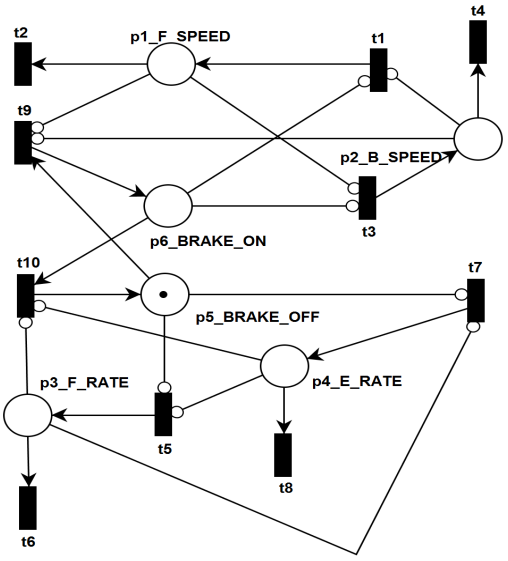
Table 7.5: Garbage system commands and sensors definition

Control variable	**Unit**	**Comment**
F-SPEED	dm/s	Forward speed
B-SPEED	dm/s	Backward speed
F-RATE	hg/s	Filling rate
E-RATE	hg/s	Emptying rate
BRAKE-ON	unit less	Apply brake
BRAKE-OFF	unit less	Release brake

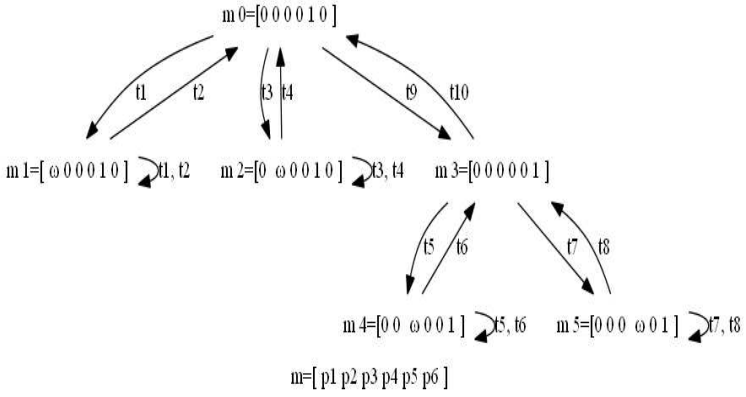
Sensor Variable	**Unit**	**Comment**
pos	m	Position sensor
w_1	kg	Discharge container weight
w_2	kg	Transport container weight
w_3	kg	Remaining garbage weight

7.7.3 Processes and goals

After having defined the basic control of the transport container, we need do design a set of *reactive processes*. As mentioned earlier a *reactive process* has a goal, triggering and exiting conditions, and a



(a)



(b)

Figure 7.12: MICPTI net of garbage system and its characteristic

priority. In Table 7.6 nine processes are defined using three priority levels. The convention is that the higher the priority is, the more important is the process. The processes $proc_1$ and $proc_3$ are to be seen as anti collision processes. The process $proc_1$ triggers when the position is almost 30 meters (maximum), and upon triggering

it aims at realising the goal g_1, which sets the F-SPEED=0. The process $proc_3$ is associated with the g_3 and triggers when the position approaches 0 meters (the minimum). The processes $proc_2$ and $proc_4$ trigger to either move the transport container forward or backward. Their associated goals g_2 and g_4 specify the required forward or backward speeds respectively. The process $proc_5$ triggers when the job is done, upon triggering it applies the brake as specified by g_5. The processes $proc_6$ and $proc_9$ trigger to either empty or fill the transport container, and are conditioned by the position of the transport container and its weight. For $proc_9$ the remaining weight is used as a condition to trigger filling. The emptying rate is specified by g_6 which is set to 5 hg/s, while the filling rate is specified by g_9 and is set to 4 hg/s. Finally, $proc_7$ and $proc_8$ trigger to interrupt the emptying and the filling operations respectively.

7.7.4 Simulation result

In this simulation we run the nine processes defined above concurrently, and adopt a priority preemptive process scheduling strategy (see Algorithm 18). The initial parameters are the following:

- The initial positions is zero m ($pos = 0$)
- The discharge weight is zero kg $w_1 = 0$
- The transport weight is zero kg $w_2 = 0$
- The remaining weight is 120 kg $w_3 = 120$ kg

Figure 7.13a shows three plots. The first one describes which and when a processes runs. The second shows position changes, and the third shows the weight changes. We can clearly see that the transport container moves between 0 and 30 m. Moving the container from 0 to 30 meters is justified by the triggering of $proc_2$, while the backward movement is justified by the triggering of $proc_4$. We can also see that $proc_1$ triggers when the position is almost 30 m, and that $proc_3$ triggers when the position is almost 0 m. As mentioned earlier these processes act as collision prevention. The weight change is also clearly correlated with the triggering of $proc_9$ for filling and $proc_6$ for emptying the container.

These two processes are immediately followed by $proc_8$ and $proc_7$ to avoid that the container gets over filled, or emptied when its

Table 7.6: Garbage system process definitions and goals relations

Proc.	**Goal**	**Trigger cond.**	**Exit cond.**	**Pri.**
$proc_1$	g_1	$tc_1 = pos > 29$	$ec_1 \neq tc_1$	2
$proc_2$	g_2	$tc_2 = pos < 29$	$ec_2 \neq tc_2$	1
$proc_3$	g_3	$tc_3 = pos < 1$	$ec_3 \neq tc_3$	2
$proc_4$	g_4	$tc_4 = pos > 1 \wedge w_2 > 1$	$ec_4 = pos < 1$	1
$proc_5$	g_5	$tc_5 = w_3 < 1 \wedge$ $w_2 < 1 \wedge 3 < pos < 10$	$ec_5 \neq tc_5$	3
$proc_6$	g_6	$tc_6 = pos < 1 \wedge w_2 > 1$	$ec_6 = w_2 < 1$	1
$proc_7$	g_7	$tc_7 = w_2 < 1$	$ec_7 \neq tc_7$	2
$proc_8$	g_8	$tc_8 = w_2 > 49$	$ec_8 \neq tc_8$	2
$proc_9$	g_9	$tc_9 = pos > 29$ $\wedge w_2 < 49 \wedge w_3 > 1$	$ec_9 \neq tc_9$	1

Goals	**Tot Inc.**	**Part Inc.**	**Mu Inc.**	**Mu Exc.**
$g_1[m(p_1) = 0]$		$g_3, g_4, g_5, g_6, g_7, g_8, g_9$		g_2
$g_2[m(p_1) = 3]$	g_3,g_7,g_8			g_1, g_4, g_5, g_6, g_9
$g_3[m(p_2) = 0]$		$g_1, g_2, g_5, g_6, g_7, g_8, g_9$		g_4
$g_4[m(p_2) = 2]$	g_1,g_7,g_8			g_2, g_3, g_5, g_6, g_9
$g_5[m(p_6) = 1]$	g_1, g_3	g_6, g_7, g_8, g_9		g_2, g_4
$g_6[m(p_4) = 5]$	g_1, g_3, g_5, g_8			g_2, g_4, g_7, g_9
$g_7[m(p_4) = 0]$		$g_1, g_2, g_3, g_4, g_5, g_8, g_9$		g_6
$g_8[m(p_3) = 0]$		$g_1, g_2, g_3, g_4, g_5, g_6, g_7$		g_9
$g_9[m(p_3) = 4]$	g_1,g_3,g_5, g_7			g_2, g_4, g_6, g_8

is already empty. The process $proc_5$ triggers at the end of the simulation when all the garbage has been transported and discharged. Figure 7.13 shows the commands changes over time. The target commands are dotted curves marked with T in the legend, and the current commands are solid lines plots marked with C. The target commands are the result of the computed marking by the scheduling algorithm, and the current describes the actual commands that are

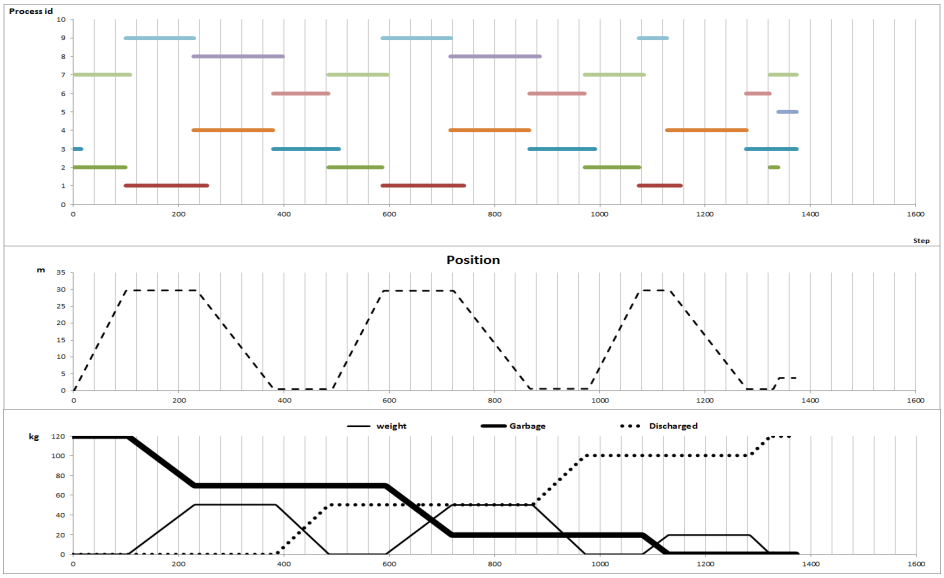
being executed. A target command is to be seen as a set point, while the current command is the intermediate steps before reaching the target set point. The guaranty that each command is legal is provided by the scheduler. The scheduler in turn bases its calculation of the target commands and their corresponding intermediate commands on the MICPTI net results. In particular the solutions that MICPTI provide for the path and reachability problems.

This experiment shows that it is possible to compose an unmanned reactive system using separated processes, where each process provides a specific competence. No global environment model was require, but only partial models, such as *stop when to close to end*, or *fill when not yet filled* etc.

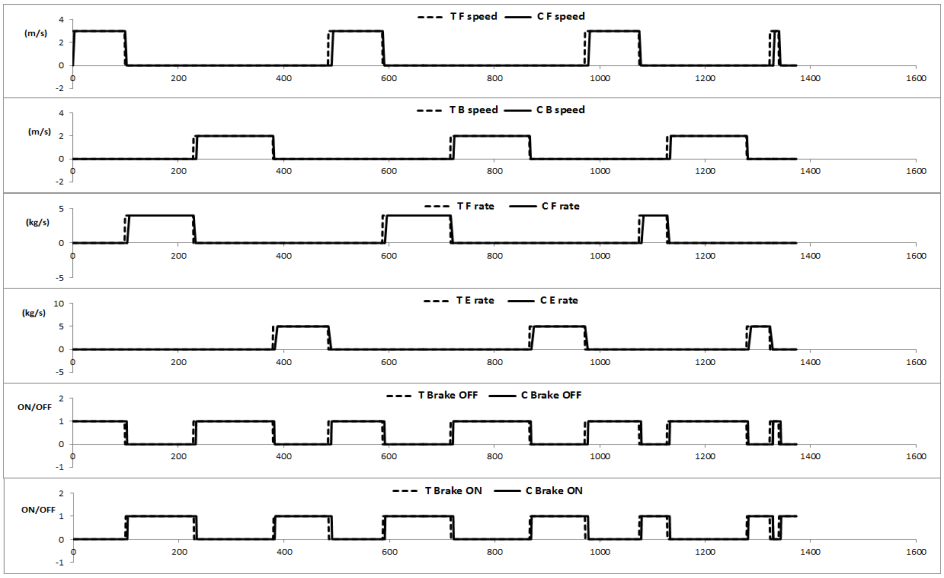
7.8 Chapter summary

At the beginning of this chapter we advocated a modelling approach that decouples the system from the environment in which it is deployed, and re-couples again by designing proper processes for specific tasks. We assumed that the system dynamic can be captured in a Petri net model in which the path and reachability problems are solvable. We have therefore used finite systems net and infinite systems. The first provides an explicit reachability graph in which the path problem is decidable, while the second is restrained to MICPTI.

These two types of nets give us the guaranty that the system dynamics is captured and can thus be verified for correctness. The same thing can not be said about the environment as we don't dispose a model that captures the complete environment dynamics. The idea is to have fragments of models and find a way to combine these fragments into a complete system. The responsibility of a *reactive process*is to encapsulate partial models, observe key parameters and trigger when conditions are present. The means by which a process reacts is constrained by the Petri net model. We call the process reaction a *goal*, which is specified over a set of places and their corresponding values. Being able to determine which states correspond to which goals gives us the possibility to identify eventual conflicting goals, and thus eventual conflicting process reactions.



(a) process executions and sensor readings changes



(b) commands changes

Figure 7.13: Simulation results

We defined four types of goal relations: mutually exclusive, mutually inclusive, partially inclusive and totally inclusive goals. These relations are important when it comes to designing a process scheduler. For clarity, given a system which is in some marking m, when a *reactive process* $proc_a$ associated to a goal g_a triggers, the scheduler has to determine a sequence of markings that leads to some marking m_a which satisfies the goal g_a. When another process that has a goal g_b triggers, it becomes important to know which relation binds g_a and g_b. This relation allows us to determine if there exists a sequence of marking from m_a to some marking m_b that satisfies g_b. We called such a path a feasible path and proposed an algorithm for it.

The scheduling problem we faced is categorised in the on-line scheduling problems. The processes arrive over time, but the scheduler does not know which process arrives next. We considered preemptive and non preemptive processes, and proposed algorithms based on two scheduling strategies: First-In-First-Out and priority scheduling.

Finally we put the things together in an illustrative simulation example, where nine processes where defined and run in parallel. The experiment has shown that it is possible to obtain a satisfactory global behaviour out of basic local behaviours. This notion is not new as such, since it is a the heart of the subsumption architecture. However the added value is on the structure we proposed when designing a subsumption based system.

7.9 Algorithms

Algorithm 11 A goal g, a marking m outputs m' satisfies g

```
1: Initialization:
2:    CG ← (V, E)                          {Characteristic graph}
3:    Mg ← ∅                               {An empty set of markings satisfying g}
4: function Satisfy(g, CG)
5:    for all m ∈ CG
6:       m' ← Project(g, m)
7:       if m' ≠ NIL
8:          Mg ← Mg ∪ {m'}
9:    return Mg                            {Return the set of marking which satisfy g}
10: function Project(g, m)
11:   m' ← m
12:   for all p ∈ P_g
13:      if g[m(p)] ≠ m'(p)
14:         if m'(p) ≠ ω
15:            return NIL
16:         else
17:            m'(p) ← g(p)
18:   return m'                            {Return the projection of m that matches g}
```

Algorithm 12 Determine whether $A \subseteq B$ A and B are sets of markings

```
1: function Contains(A, B)
2:   for all m ∈ A
3:     for all m' ∈ B
4:       if Covers(m, m') ≠ TRUE
5:         return FALSE
6:   return TRUE
7: function Covers(m, m')
8:   for all p ∈ P
9:     if m(p) ≠ m'(p) ∧ m'(p) ≠ ω
10:      return FALSE
11:  return TRUE
```

Algorithm 13 Determine $A \cap B$, where A and B are sets of markings

```
1: function Intersection(A, B)
2:   I ← ∅                                   {Intersection}
3:   for all m_a ∈ A
4:     for all m_b ∈ B
5:       m' ← Match(m_a, m_b)
6:       if m' ≠ NIL
7:         I ← I ∪ {m'}
8:   return I
9: function Match(m_a, m_b)
10:  m' ← NEW
11:  for all p ∈ P
12:    if m_a(p) ≠ m_b(p) ∧ m_a(p) ≠ ω ∧ m_b(p) ≠ ω
13:      return NIL
14:    else
15:      m'(p) = MIN(m_a(p), m_b(p))
16:  return m'
```

Algorithm 14 Determine $\bigcap \Theta$, where Θ is a set of sets of markings

```
1: function CumIntersection(Θ)
2:   CI ← ∅
3:   if Θ = ∅
4:     return CI
5:   else
6:     CI ← Θ[0]                          {CI takes the first set in Θ}
7:   if length(Θ) > 1
8:     for i = 1 to length (Θ)-1
9:       CI ← Intersection(CI, Θ[i])
10:      i ← i + 1
11:  return CI
```

Algorithm 15 Feasible path between a goal g and a set of goals G_m that are satisfied by a marking m

```
1: function feasiblepath(g_a, G_m, m, (V, E))
2:   P ← NIL
3:   Mg_a ← Satisfy(g_a, (V, E))
4:   θ ← ∅
5:   for all g ∈ G_m
6:     θ ← θ ∪ Satisfy(g, (V, E))
7:   if θ = ∅
8:     for all m_a ∈ Mg_a
9:       P ← Path(m, m_a, (V, E))
10:      if P ≠ NIL
11:        break
12:  else
13:    I ← CumIntersection(θ)
14:    for all m_a ∈ Mg_a
15:      P ← Path(m, m_a, (V, E))
16:      if P ≠ NIL ∧ P ⊆ I
17:        break
18:  return P
```

Algorithm 16 Deciding whether a reactive process runs or waits

```
1: function Insert(w, R, Q_tmp, m, G)
2:    done ← FALSE
3:    for all r ∈ R
4:       if w ∈ MEX(r)
5:          Q_tmp.enqueue(w)
6:          done ← TRUE
7:          break
8:       if w ∈ MINC(r) ∨ w ∈ TINC(r)
9:          R ← R ∪ {w}
10:         done ← TRUE
11:         break
12:   if done ≠ TRUE
13:      P ← feasiblepath(w, R, m, G)
14:      if P ≠ NIL
15:         m ← execute(P)
16:         R ← R ∪ {w}
17:      else
18:         Q_tmp.enqueue(w)
```

Algorithm 17 FIFO Scheduling Algorithms

```
1: function NonPreemptiveFIFO(Q_w, R, m, G)
2:     Q_tmp ← ∅
3:     while Q_w ≠ ∅
4:         w ← Q_w.dequeue()
5:         Insert(w, R, Q_tmp, m, G)
6:     Q_w ← Q_tmp
7: function PreemptiveFIFO(Q_w, R, m, G)
8:     Q_tmp ← ∅
9:     while Q_w ≠ ∅
10:        w ← Q_w.dequeue()
11:        for all p ∈ R
12:            if w.trigger < p.trigger
13:                R ← R − {p}
14:                Q_w.Enqueue(p)
15:        Insert(w, R, Q_tmp, m, G)
16:    Q_w ← Q_tmp
```

Algorithm 18 Priority based Scheduling Algorithms

```
1: function NonPreemptivePriority(Q_w, R, m, G)
2:     Q_tmp ← ∅
3:     Q_w ← Q_w.priositySort()
4:     while Q_w ≠ ∅
5:         w ← Q_w.dequeue()
6:         Insert(w, R, Q_tmp, m, G)
7:     Q_w ← Q_tmp
8: function PreemptivePriority(Q_w, R, m, G)
9:     Q_tmp ← ∅
10:    Q_w ← Q_w.priositySort()
11:    while Q_w ≠ ∅
12:        w ← Q_w.dequeue()
13:        for all p ∈ R
14:            if w.priorty() > p.priorty()
15:                R ← R − {p}
16:                Q_w.priorityEnqueue(p)
17:        Insert(w, R, Q_tmp, m, G)
18:    Q_w ← Q_tmp
```

Chapter 8

The Software Packages

In this chapter we present two software packages: A *Petri nets analysis* package implementing the theory of Chapter 6, and a *Goal analysis* package implementing the theory of Chapter 7.

8.1 Introduction

Today, there are 81 registered Petri net tools and software libraries [59], each with distinct benefits and drawbacks with respect to their features and user friendliness. However, exiting tools do not address the CPTI and MICPTI classes which were introduced in Chapter 6, nor do they address the theory of Chapter 7.

For these reasons, we introduce two C# packages: One for the analysis of CPTI, and MICPTI nets, and the other for the analysis of *goals*. We will show using lines of code how to perform the different analyses for determining the properties of nets like CPTI and MICPTI. We also show how to specify *goals* and compute the four relations from Chapter 7,i.e. *Mutual Exclusion*, *Mutual Inclusion*, *Total Inclusion* and *Partial Inclusion*.

Figure 8.1 gives an overview of the architecture. The figure shows a Petri net editor that is used to draw Petri net models. The features of the editor are outside the scope of this thesis. However, among the many Petri nets tools [59] we have chosen an editor called Platform Independent Petri net Editor version 2.5 (PIPE) [18], which we found

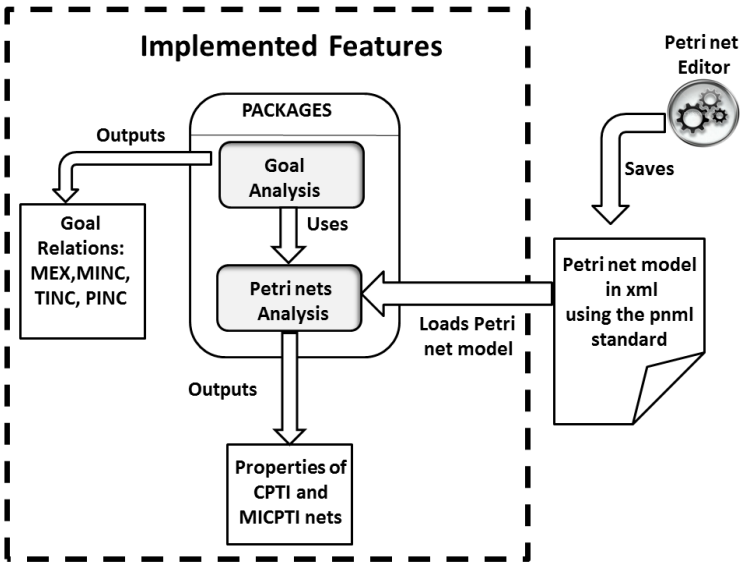


Figure 8.1: An overview of the simulation tool

both stable and user friendly. Using PIPE one can draw and simulate nets, allowing the user to have an idea about the expected behaviour of the modelled net.

The Petri net model is then saved in the Petri net Mark-up Language format (PNML) [16]. PNML is an XML based standard kept by the Petri net community, to ease the interoperability between different Petri net tools.

Using our *Petri net Analysis* package, it is possible to load Petri nets in PNML and analyse them further. The main purpose of this package is to analyse the CPTI and MICPTI types of nets from Chapter 6. The use of this package will be addressed in Section 8.3 and Section 8.4.

The *Goal Analysis* package is build on top of the *Petri net Analysis* package. The main purpose of the *Goal Analysis* package is to analyse the *goals* of *reactive processes* from Chapter 7. We will show how this package works in Section 8.5.

8.2 Basics

In this section we show some basic code to make the user familiar with our Petri net library.

Table 8.1 shows how to load a Petri net model from a PNML file. Once the model loaded, it is stored in an instance of the class PN (stands for Petri net). All the methods that we will use to analyse a Petri net model are called from a PN object (instance of the class PN).

For example, the actual marking of the Petri net model is obtained by calling the property method *petrinet.Marking*. One can also obtain all the enabled transitions from a given marking by calling the *petrinet.EnabledTransitions*. Firing a transition is done by calling the method *Fire(Transition t)*. The user can also specify or keep a marking and then set it as the nets current marking. This is done using the method *SetMarking(Marking m)*.

Table 8.2 shows a program for a random *token game*. This works as follows: Starting from the initial marking, we repetitively choose a randomly transition among the enabled ones and fire it. The program stops only if a marking is reached from which no transition can fire (a deadlock marking).

Table 8.1: Some basic methods from our Petri net library

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
public class BasicDemo
{
    public static void Main(string[] args)
    {
     string fileLocation = "..\\yourModel.pnml";
     PN petrinet = PNLoader.LoadPNML(fileLocation);
     //Getting the current marking
     Marking intitalMarking = petrinet.Marking;
     //Printing the current marking
     intitalMarking.print();
     //Enabled transitions from current marking
     foreach(Transition t in petrinet.EnabledTransitions)
     {
      Console.WriteLine(t.Name);
     }
     //Fire the first enable transition,
     //and print the resulting marking
     List<Transition> enabledTrans =
     petrinet.EnabledTransitions;
     if (enabledTrans != null && enabledTrans.Count > 0)
     {
      Transition trans = petrinet.EnabledTransitions[0];
      petrinet.Fire(trans);
      petrinet.Marking.print();
     }
     //set back the initial marking
     petrinet.SetMarking(intitalMarking);
     petrinet.Marking.print();
    }
}
```

Table 8.2: A random token game program

```
using System;
using System.Collections.Generic;
using Math;
using PetrinetAnalysis;
                        //Random Token Game
public class TokenGame
{
    public static void Main(string[] args)
    {
      string fileLocation = "..\\yourModel.pnml";
      PN petrinet = PNLoader.LoadPNML(fileLocation);
      Random randomNumber = new Random();
      int count = petrinet.EnabledTransitions.Count;
      while (count > 0)
      {
        //find a random number less than count
        int random =randomNumber.Next(count);
        Transition t=petrinet.EnabledTransitions[random];
        petrinet.Fire(t);
        count = petrinet.EnabledTransitions.Count;
      }
      Console.WriteLine("Random Token Game Finished:");
      petrinet.Marking.print();
    }
}
```

8.3 Code For CPTI Analyses

In Chapter 6 we presented two classes of nets: The CPTI nets, and the MICPTI nets. This section shows how to use our library to analyse CPTI nets, whereas the MICPTI class will be addressed in Section 8.4.

CPTI nets are analysed by means of their coverability graphs, however the Coverability graph algorithm 5 from Chapter 6 applies to CPTI and MICPTI nets only. This means that before running Algorithm 5 we have to make sure that the net in question is either CPTI or MICPTI.

Table 8.3 shows how to check whether a Petri is CPTI using the method *isCPTI()*, which returns a boolean value. To obtain the sets of *circular elementary structures* (ces) and *flate elementary structures* (fes), we call the methods *FlateStrutures()* and *CircularStrutures()* respectively.

Table 8.3: A program for checking whether a net is CPTI

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
                    //CPTI Check?
public class CheckingCPTI
{
  public static void Main(string[] args)
   {
    string fileLocation = "..\\yourModel.pnml";
    PN petrinet = PNLoader.LoadPNML(fileLocation);
    bool iscpti=petrinet.isCPTI();
    Console.WriteLine("The net is CPTI: " + iscpti);
    //Get the sets of CES and FES
    if(iscpti)
     {
      List<FES> Set_fes = petrinet.FlateStrutures();
      List<CES> Set_ces = petrinet.CircularStrutures();
     }
   }
}
```

8.3.1 Coverability Graph

To obtain the coverability graph of a CPTI net, we call the method *CPTI_ MICPTI_ Coverability_ Graph()* which returns a list of markings as shown in Table 8.4. In our implementation, markings are also nodes of graph. That is, a marking can have one or several successor markings.

Finally, to visualize the coverability graph we use the Dot language [52], which can be visualized using the Graphviz tool [40]. By calling the method *Save(List< Marking > somegraph)* a dot file is generated, which in turn can be loaded using Graphviz.

8.3.2 Boundedness

A CPTI net is bounded if its coverability graph contains no extended marking. When a CPTI net is unbounded, it means that there exist one or several places that could contain infinitely many tokens. Table 8.5 shows how to check the boundedness of a CPTI net and how to obtain the set of unbounded places.

8.3.3 Deadlock

A marking is deadlock if no transition can fire from it. Table 8.6 shows how to get the set of deadlock markings.

8.3.4 Transition Liveness

For a CPTI net it is possible to determine whether a transition is quasi-live or dead. A transition is dead if it is never enabled. A transition is quasi-live if there exists a marking from which it can fire. A transition which is not dead is necessarily quasi-live. Table 8.7 shows how to get the set of dead transitions.

Table 8.4: Getting the coverability graph of a CPTI net

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
public class CPTICoverabilityGraph
{
   public static void Main(string[] args)
   {
    string fileLocation = "..\\yourModel.pnml";
    PN petrinet = PNLoader.LoadPNML(fileLocation);
    bool iscpti = petrinet.isCPTI();
    if (iscpti)
    {
     List<Marking>coverabilityGraph=
     petrinet.CPTI_MICPTI_Coverability_Graph();
     if(coverabilityGraph!= null &&
       coverabilityGraph.Count > 0)
         {
     Marking m = coverabilityGraph[0];
     List<Connection> links = m.Connections;
       foreach (Connection c in links)
          {
         Marking successor_of_m = c.To;
          }
         }
     petrinet.Save(coverabilityGraph, "...file.dot");
     }    }  }
```

Table 8.5: A program for checking the boundedness of a CPTI net

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
            //Boundedness checking
public class CPTIBoundedness
{
 public static void Main(string[] args)
   {
    string fileLocation = "..\\yourModel.pnml";
    PN petrinet = PNLoader.LoadPNML(fileLocation);
    bool iscpti = petrinet.isCPTI();
   if (iscpti)
    {
    //Get the coverability graph
    List<Marking> coverabilityGraph =
    petrinet.CPTI_MICPTI_Coverability_Graph();
    //is the net bounded?
bool bounded = petrinet.isBounded(coverabilityGraph);
    //list of unbounded places
    List<Place> unboundedPlaces =
 petrinet.GetUnboundedPlaces(coverabilityGraph);
      }
  }
}
```

Table 8.6: A program for getting deadlock markings

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
                //Deadlock checking
public class CPTIDeadlock
{
  public static void Main(string[] args)
  {
   string fileLocation = "..\\yourModel.pnml";
   PN petrinet = PNLoader.LoadPNML(fileLocation);
   bool iscpti = petrinet.isCPTI();
   if (iscpti)
   {
   //Get the coverability graph
  List<Marking> coverabilityGraph =
  petrinet.CPTI_MICPTI_Coverability_Graph();
  //list of deadlock markings
 List<Marking> deadlocks =
 petrinet.GetDeadlockMarkings(coverabilityGraph);
        }
    }
}
```

Table 8.7: A program for getting dead transitions

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
            //Dead transitions
public class TransitionLiveness
{
  public static void Main(string[] args)
  {
   string fileLocation = "..\\yourModel.pnml";
   PN petrinet = PNLoader.LoadPNML(fileLocation);
   bool iscpti = petrinet.isCPTI();
   if (iscpti)
     {
      //Get the coverability graph
       List<Marking> coverabilityGraph =
       petrinet.CPTI_MICPTI_Coverability_Graph();
       //list of dead transitions
       List<Transition> deadTransition =
       petrinet.GetDeadTransitions(coverabilityGraph);
     }
  }
}
```

8.4 Code For MICPTI Analyses

In Chapter 6, we defined MICPTI as a subclass of CPTI. This means that the lines of code from Section 8.3 can also be used to analyse MICPTI nets. However, we can determine additional properties for MICPTI nets such as: Marking reachability, transition liveness (not only quasi-liveness), reversibility of a net and finding paths from a marking to another (see Table 8.8).

Table 8.8: A program for checking whether a net is MICPTI

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
//Checking if a net is MICPTI
public class CheckingMICPTI
{
    public static void Main(string[] args)
    {
      string fileLocation = "..\\yourModel.pnml";
      PN petrinet = PNLoader.LoadPNML(fileLocation);
      bool isMICPTI = petrinet.isMICPTI();
      Console.WriteLine("The net is MICPTI: " + isMICPTI);
      if (isMICPTI)
       {
         // MICPTI analysis comes here...
       }
    }
}
```

8.4.1 Marking Reachability

We mentioned in Chapter 6 that it is possible to determine the reachability of a marking for MICPTI nets. Table 8.9 shows how to use our library to determine the marking reachability for MICPTI nets.

Table 8.9: A program for checking the reachability of a marking

```
using System;
using System.Collections.Generic;
using Math;
using PetrinetAnalysis;
    //Checking whether a marking is reachable
public class MarkingReachability
{
  public static void Main(string[] args)
   {
    string fileLocation = "..\\yourModel.pnml";
    PN petrinet = PNLoader.LoadPNML(fileLocation);
    bool isMICPTI = petrinet.isMICPTI();
    Random random=new Random();
    if (isMICPTI)
     {
     List<Marking>coverabilityGraph=
     petrinet.CPTI_MICPTI_Coverability_Graph();
     //copying the net's marking
     Marking mToCheck = petrinet.Marking.Copy();
     //modify the mToCheck with random values
     for (int i = 0; i < mToCheck.Counts.Length; i++)
      {
      mToCheck.Counts[i] = random.Next();
      }
     //check if mToCheck is reachable
bool isReachable =
petrinet.isReachable(mToCheck, coverabilityGraph);
     }
   }
}
```

8.4.2 Characteristic Graph

In Chapter 6 Section 6.10, we introduced a so called characteristic graph. This graph is generated from the coverabililty graph to encode additional information about the connections between markings.

Table 8.10 shows how to obtain the characteristics graph of a given coverability graph, which can also be saved, and later on visualized

using Graphviz (as shown previously in Table 8.4).

Table 8.10: A program for obtaining the characteristic graph

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
//Getting the Characteristic Graph
public class CharGraphMICPTI
{
  public static void Main(string[] args)
  {
   string fileLocation = "..\\yourModel.pnml";
   PN petrinet = PNLoader.LoadPNML(fileLocation);
   bool isMICPTI = petrinet.isMICPTI();
   if (isMICPTI)
     {
      List<Marking> coverabilityGraph =
      petrinet.CPTI_MICPTI_Coverability_Graph();
      List<Marking> characteristicGraph =
      petrinet.GetCharacteristicGraph(coverabilityGraph);
      //Saving the Graph in the language dot
      //Dot is used by Graphviz to visualize graphs.
      string savingLocation = "...file.dot";
      petrinet.Save(characteristicGraph, savingLocation);
     }
  }
}
```

8.4.3 Reversibility

It is possible to check whether a MICPTI is reversible using its characteristic graph. As discussed in Section 6.10, a net is reversible if its characteristic graph constitutes one strongly connected component. A consequence of the reversibility of a net is that a quasi-live transition is guaranteed to be live. Table 8.11 shows how to determine the reversibility of a MICPTI net.

Table 8.11: A program for checking whether a MICPTI net is reversible

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis;
//Checking whether a net is reversible
public class MICPTIReversible
{
 public static void Main(string[] args)
  {
   string fileLocation = "..\\yourModel.pnml";
   PN petrinet = PNLoader.LoadPNML(fileLocation);
   bool isMICPTI = petrinet.isMICPTI();
   if (isMICPTI)
    {
    List<Marking> coverabilityGraph =
    petrinet.CPTI_MICPTI_Coverability_Graph();
    List<Marking> characteristicGraph =
    petrinet.GetCharacteristicGraph(coverabilityGraph);
    bool isReversible =
    petrinet.IsReversible(characteristicGraph);
    }
  }
}
```

8.4.4 Finding Paths

In Section 6.10 we presented and algorithm for determining a sequence of firings from one marking to another. The code in Table 8.12 chooses two random reachable markings, and finds a path from the one to other.

Table 8.12: A program for finding a path between two markings

```
using System;
using Math;
using System.Collections.Generic;
using PetrinetAnalysis;
//Find a path between two reachable markings
public class MICPTIPath
{
  public static void Main(string[] args)
  {
  string fileLocation = "..\\yourModel.pnml";
  PN petrinet = PNLoader.LoadPNML(fileLocation);
  bool isMICPTI = petrinet.isMICPTI();
  if (isMICPTI)
  {
   List<Marking> coverabilityGraph =
   petrinet.CPTI_MICPTI_Coverability_Graph();
   Marking start =
   petrinet.GetRandomMarking(coverabilityGraph);
   Marking destination =
   petrinet.GetRandomMarking(coverabilityGraph);
   List<Transition> path=
   petrinet.FindPath_In_MICPTI
   (start, destination, coverabilityGraph);
   }
  }
}
```

8.5 Code For Goal Analyses

The *GoalAnalysis* package implements the approach that was presented in Chapter 7. More precisely, using this package one can determine four relations between goals: The mutual inclusion, mutual exclusion, partial inclusion, and total inclusion.

To analyse the relation between different goals using this package, one roughly needs to do following: Load a MICPTI net, specify a set of goals, and call a method *AnalyseGoals*. To show how this can be done we use the MICPTI net of Figure 8.2, which was used earlier in Section 7.7.

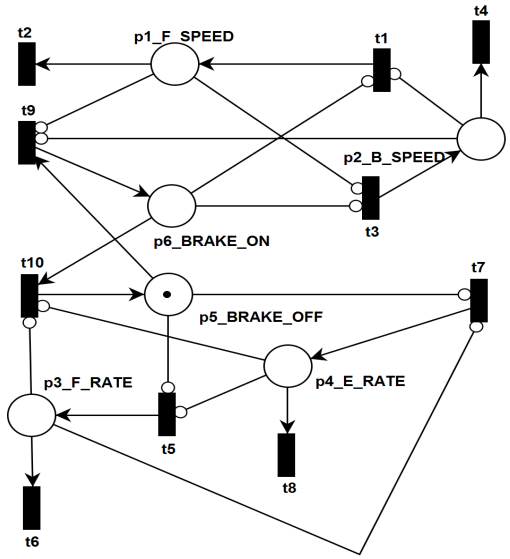


Figure 8.2: MICPTI net of the garbage system from Section 7.7

```
using System;
using System.Collections.Generic;
using PetrinetAnalysis
using GoalAnalysis;
public class AnalysingGoals
{
  public static void Main(string[] args)
  {
    string fileLocation = "..\\Figure7.12.pnml";
```

```
10      PN petrinet = PNLoader.LoadPNML(fileLocation);
11      List<Goal> goals = new List<Goal>();
12      //goal name
13      Goal g1 = new Goal("g1");
14      Place place=petrinet.GetPlace("p1_F_SPEED");
15      g1.Specify(place, 0);
16      Goal g2 = new Goal("g2");
17      g2.Specify(place, 3);
18      Goal g3 = new Goal("g3");
19      place=petrinet.GetPlace("p2_B_SPEED");
20      g3.Specify(place, 0);
21      Goal g4 = new Goal("g4");
22      g4.Specify(place, 2);
23      Goal g5 = new Goal("g5");
24      place = petrinet.GetPlace("p6_BRAKE_ON");
25      g5.Specify(place, 1);
26      Goal g6 = new Goal("g6");
27      place = petrinet.GetPlace("p4_E_RATE");
28      g6.Specify(place, 5);
29      Goal g7 = new Goal("g7");
30      g7.Specify(place, 0);
31      Goal g8 = new Goal("g8");
32      place = petrinet.GetPlace("p3_F_RATE");
33      g8.Specify(place, 0);
34      Goal g9 = new Goal("g9");
35      place = petrinet.GetPlace("p3_F_RATE");
36      g9.Specify(place, 4);
37      goals.AddRange(new Goal[]{g1,g2,g3,g4,g5,g6,g7,g8,g9});
38
39      MICPTIGoalAnalyser analyser = new MICPTIGoalAnalyser();
40      analyser.AnalyseGoals(goals, petrinet);
41
42      //Results
43      foreach(Goal g in goals)
44        {
45          List<Goal> mutualExclusive=g.MEX ;
46          List<Goal> mutualInclusive = g.MINC;
47          List<Goal> partialInclusive = g.PINC;
48          List<Goal> totalInclusive = g.TINC;
49        }
50      }
51  }
```

8.6 Chapter Summary

In this chapter we presented two C# software packages: a *Petri nets Analysis* and a *Goal Analysis* package. Those to packages implement our developed theory from Chapter 6 and Chapter 7 receptively.

Part III

Application

Chapter 9

Drilling Control System

In Chapter 3, we mentioned two major limitations of existing drilling control systems: They lack supervisory control, and fail to trigger responses to incidents. To cope with those limitations we suggest two additional components: a *command controller*, and a *safety process scheduler*. This chapter shows how to realize those two components using the theory from Chapters 5, 6 and 7.

9.1 Introduction

In Chapter 3 we suggested to improve existing drilling control by adding two components: A *command controller*, and a *safety process scheduler* as shown in Figure 9.1. The *command controller* acts as a control supervisor to limit the driller's actions to only those permissible, while the *safety process scheduler* enables the composition with reactive processes.

Realizing the *command controller* can be assimilated to the supervisory control problem [25, 74]. This is done by imposing a control supervisor to an already unsatisfactory rig control. In other words, the rig offers a multitude of possibilities, but not all of them should be allowed. Restricting the possibilities to only those that satisfy the specifications is the responsibility of the supervisor. In practice, the role of the *command controller*, is to make sure that any control variable that is sent to the *command interface* is legal.

Realizing the *safety process scheduler* takes its foundation from chapter 7. Reactive processes observe the well dynamics through sensory readings and generate appropriate responses when incidents occur. The *safety process scheduler* needs to schedule those processes, and interact with the *command controller*. We divide the rig

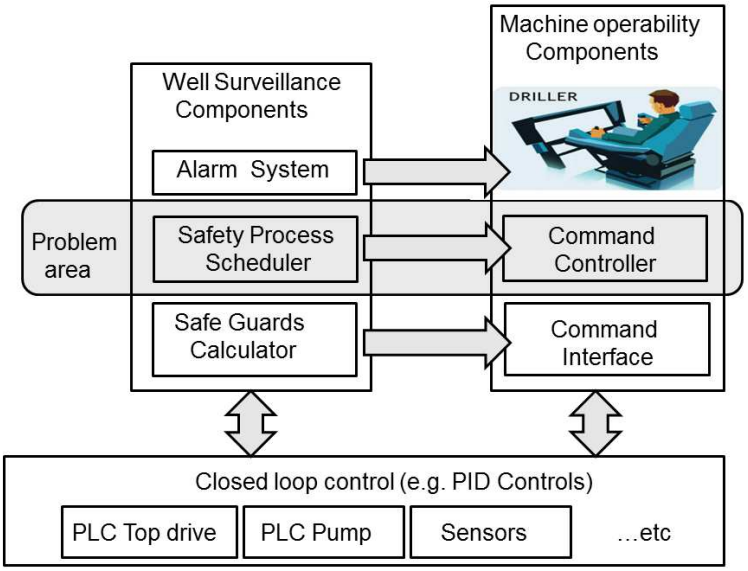


Figure 9.1: Two components

operability into two modes: A pipe handling mode and an operational mode. In the pipe handling mode the driller's main concerns is to connect or disconnect a drill-pipe to the drill-string. While in the operational mode, the operator is mainly concerned with drilling, reaming, hole-cleaning, tripping-in, tripping out etc. That is, in the operational mode the operations involved are those which could influence the well state, which is not the case for the pipe handling mode.

In addition, in the pipe handling mode the mud pump and top-drive are not used, and the draw-works is used to position the elevator (device used to hold the drill-pipe, or drill-string) either at the top (crown block) or at bottom (drill-floor). This simplifies the movement of the drill-string as we don't need to model the complete movement, but only use two discrete positions (top and bottom).

On the other hand, the drill-string movement under the operational system is modelled using velocity control, which means that the drill-string head can be at any position between the crown-block and the drill-floor. However, both modes need a *command controller*.

Assumptions

Control variables are assigned and sent to the different devices via the command interface. We assume that those control variables will be executed after a period of time. This means that individual rig devices are not considered faulty. We call the whole set of control variables assignments for the machine state.

We also assume the *command controller* is the only one that can change the machine state. This is not the case for the well state, which we assume can change at any time.

9.2 The Pipe Handling Mode

The Petri net model capturing the pipe handling mode uses five drilling tools. To each tool we associate a set of places and transitions, as shown in Table 9.1. The places will be interpreted as control variables, and the transitions as the actions taken by a driller. In addition we will use the following four places as internal variables:

- pipe_at_rack_int: This place is used to describe that a drill-pipe is actually at the star-racker.
- pipe_at_wc_int: This place indicates whether a drill-pipe is placed at the well center or not.
- string_at_bottom_int: This place indicates whether the drill-string is at the drill-floor level or not.
- string_at_top_int: This place indicates whether the drill-string is at a high position or not.

The complete model of pipe handling mode is illustrated in Figure 9.2. As the figure shows the model is hard to read, we will therefore divide it into five sub-models that are easier to understand.

Table 9.1: Pipe handling events and control variables

Tools	**Places**	**Transitions**
Draw-works	DW_EL_at_top, DW_EL_at_bottom	DW_string_to_top, DW_string_to_bottom, DW_EL_to_bottom, DW_EL_to_top
ELevator	EL_hold, EL_release	EL_hold_at_bottom, EL_hold_at_top, EL_release_at_bottom, EL_release_at_top, EL_hold_new_pipe, EL_release_pipe
Rack-arm	RA_at_wc, RA_at_rack, RA_hold, RA_release	RA_to_wc,RA_to_rack, RA_hold_pipe_at_wc, RA_hold_pipe_at_rack, RA_release_at_wc, RA_release_at_rack, RA_move_pipe_to_wc, RA_move_pipe_to_rack
Iron-roughneck	RN_idle, RN_break_off, RN_makeup	RN_break_off, RN_done_break_off, RN_makup, RN_done_makeup
Power-Slips	slips_on_, slips_off_	slips_on, slips_off

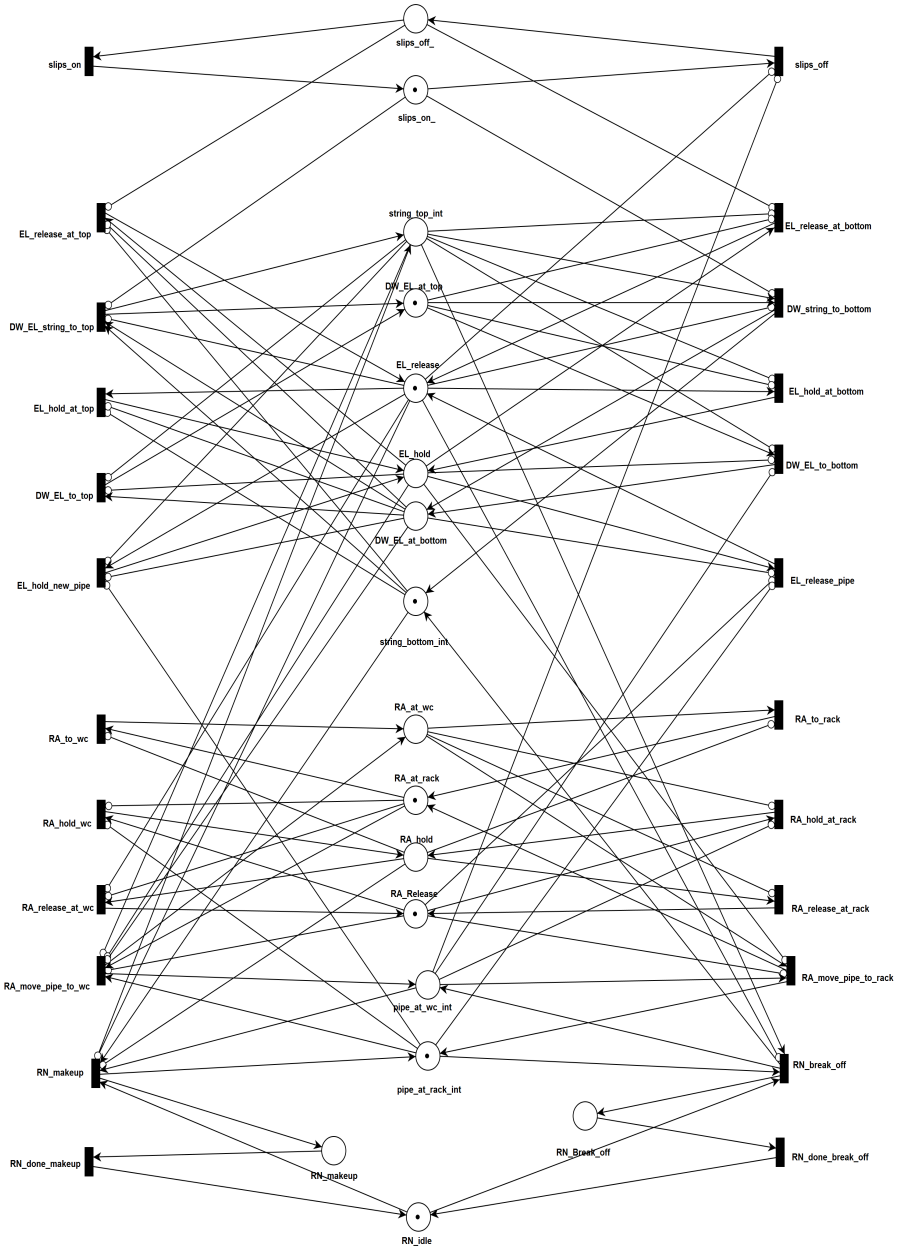


Figure 9.2: Pipe Handling model

9.2.1 The Power-Slips

Recall that the power-slips are used to suspend the drill-string in order to prevent its from falling into the well. The model in Figure 9.3 describes the power-slips dynamics. The transitions slips_off and slips_on are used to set tokens in the places slips_off_ and slips_on_. Releasing the slips requires the following:

- The slips were on
- The elevator is not on a release mode (holding the drill-string)
- The drill-pipe is not at the well center.

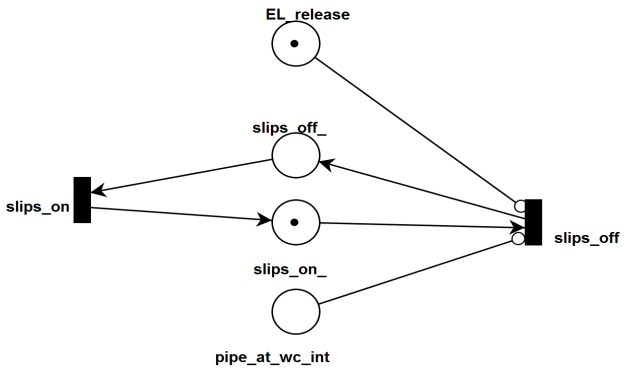


Figure 9.3: Power-slips Petri net model

9.2.2 The Elevator

The elevator is used to hold either the drill-string or the drill-pipe, when the top-drive is not connected. The Elevator model is shown in Figure 9.4. The elevator can either hold or release using the places: EL_release and EL_hold. However, in order to set tokens in those places, other conditions must be satisfied. The transitions on the net represent what can be done with the elevator and under which conditions a transition can fire. For example, releasing the elevator at the drill_floor level (transition: EL_release_at_bottom) requires that the power-slips were activated.

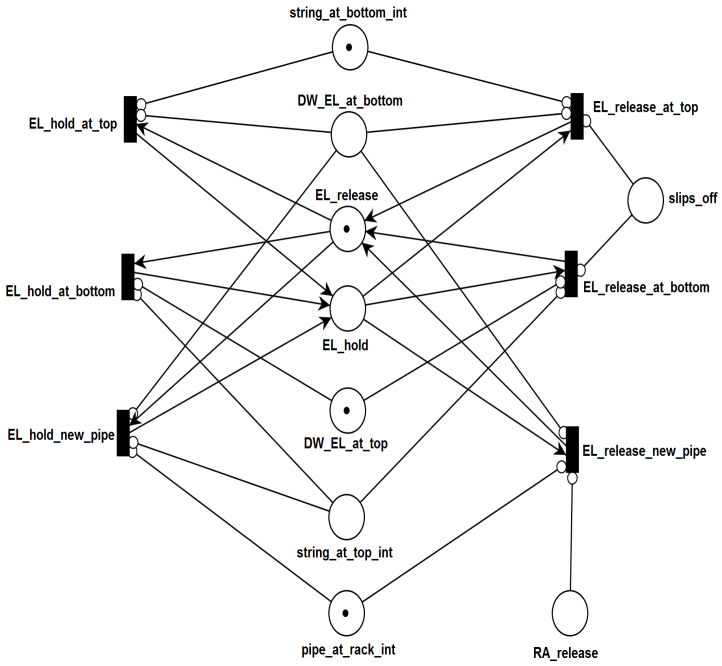


Figure 9.4: Elevator Petri net model

9.2.3 The Draw-works

The draw-works is used to obtain the vertical movement of the elevator. If the elevator is holding the drill-string then moving the Elevator to a top or bottom position will also move the drill-string. The elevator needs only to be at two positions: top or bottom. All the intermediate positions are irrelevant in pipe handling mode. When the elevator is holding the drill-string and moved upward or downwards we simply update two internal places: string_at_top_int or string_at_bottom_int (see Figure 9.5).

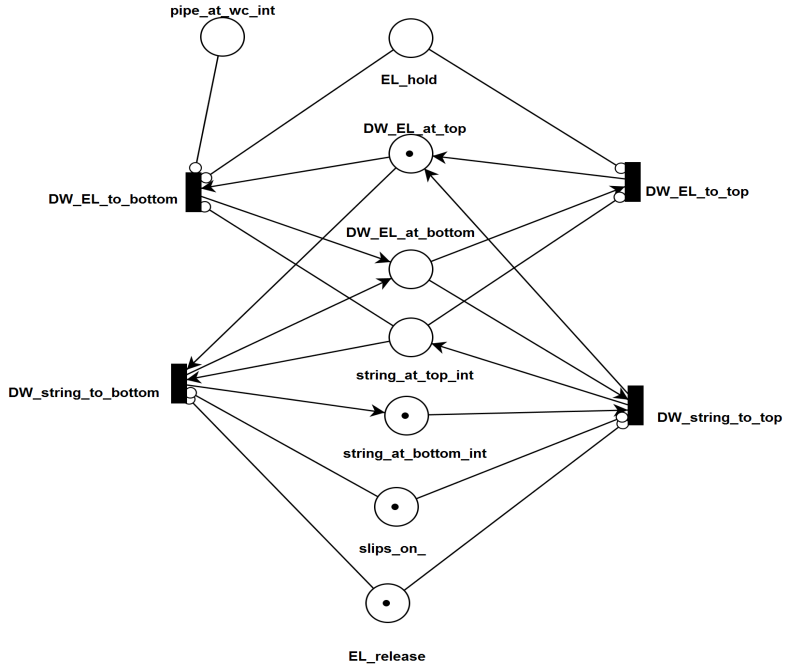


Figure 9.5: Draw-works Petri net model

9.2.4 Rack Arm

The rack arm is used to bring or deposit drill-pipes from the well-center to the star racker and vice versa. The model in Figure 9.6 shows the dynamics of the star racker arm. A typical sequence under pipe connection would be to grip a new pipe from the star racker and move it to the well-center. Before a pipe can be released, we have to make sure that the elevator is holding that pipe, otherwise it will fall on the drill-floor. When disconnecting a drill-pipe, the rack arm has to grip the pipe at the well-center and move it back to the star racker. There are eighth possible actions that a rack arm can perform, and all of them are activated under specific conditions (see Figure 9.6)

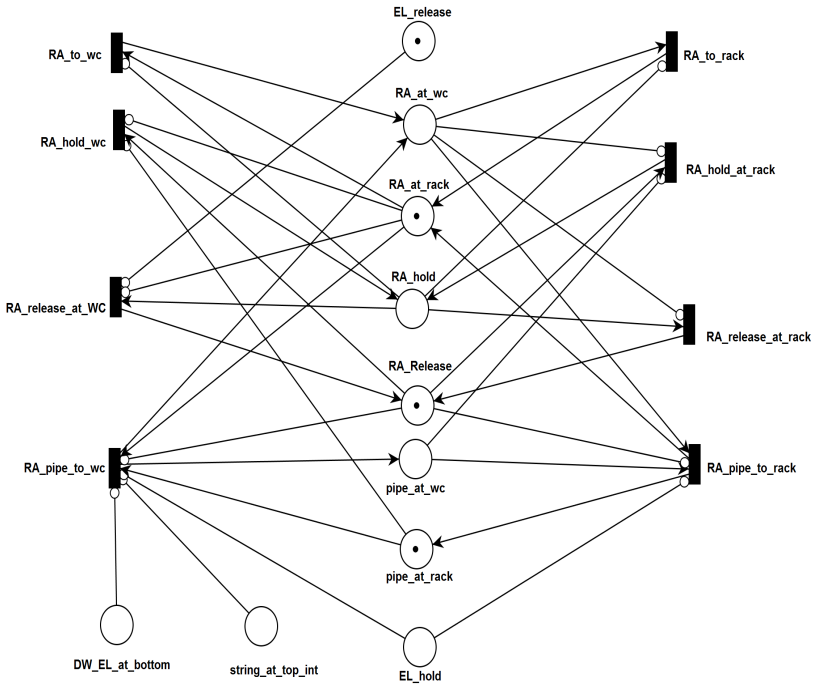


Figure 9.6: Star Racker arm Petri net model

9.2.5 Iron roughneck

The iron roughneck is used to apply the necessary make-up torque to connect a drill-pipe to the drill-string, or break-off torque to disconnect them from each others. The dynamics of the iron roughneck is shown in Figure 9.7. When connecting a new pipe we have to make sure that the pipe is at the well-center, is held by the elevator and that the drill-string is at bottom position.

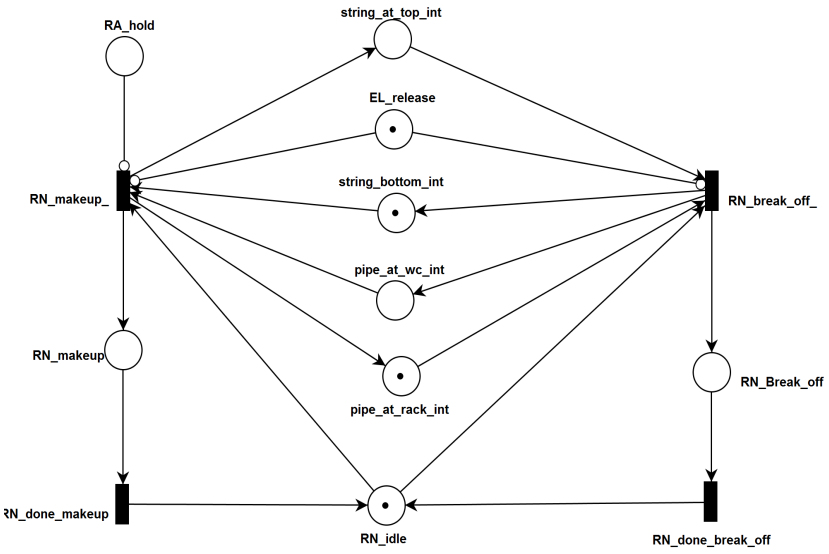


Figure 9.7: Iron-roughneck Petri net model

9.3 Analysing the Pipe Handling Model

The model in Figure 9.2 cannot be claimed correct unless it satisfies some properties. We divide these properties in three categories: General, state specific and transition specific.

9.3.1 General Properties

As there are 17 boolean variables in the pipe handling model, we have a combinatorial space of 2^{17} states. Using Algorithm 1 from

Section 5.3, we can obtain a reachability graph for the pipe handling model. The graph contains 91 states, which is sufficient to describe the legal behaviours.

The general properties of the pipe handling model are summarized in Table 9.2. Deadlock states can easily be checked by inspecting the reachability graph for states that do not contain any outgoing edges. A transition which does not appear in any edge is considered dead.

If the reachability graph constitutes a strongly connected component then the net is reversible. Checking for strongly connectedness was explained in Section 5.3.

A net is live when each of its transitions are live. Recall that from Section 5.3, a net that is quasi-live and reversible is also live. A net is bounded if its reachability graph is finite, and a net is one-safe if each place cannot contain more than one token.

The fact that the reachability graph algorithm terminates means that the net is bounded. Checking whether a net is one-safe can be done by inspecting the reachable markings. However, one-safeness can be forced by requiring that a place cannot contain more than one token.

Table 9.2: General properties of the pipe handling model

Property	**Description**
Number of states	91 states
Dead lock states	None
Dead transitions	None
Reversible	Yes
Bounded	One-safe
Liveness	Live net

9.3.2 State properties

State properties define the requirements that each state (marking) should satisfy. For example, the places (variables) slips_on_ and slips_off_ should never be equal. For that we need to specify a set of rules and check that each reachable marking satisfies those rules.

The rules in Table 9.3 are all satisfied by our model. However, when a rule is not satisfied, the model must be refined. The point is that we have a way to determine whether the captured behaviour satisfies our requirements, and thus we have the possibility to represent a system and check the correctness of that representation.

Table 9.3: State specific rules

Rule	**Formula**
1	slips_on_ $\neq$ slips_off_
2	string_at_top_int $\neq$ string_at_bottom_int
3	EL_hold $\neq$ EL_release
4	DW_EL_at_top $\neq$ DW_EL_at_bottom
5	RA_at_wc $\neq$ RA_at_rack
6	RA_hold $\neq$ RA_release
7	pipe_at_wc_int $\neq$ pipe_at_rack_int
8	RN_idel $\neq$ RN_break-off
9	RN_idel $\neq$ RN_makeup
10	$\neg$(RN_break_off $\wedge$ RN_make_up)
11	$\neg$(pipe_at_wc_int $\wedge$ string_at_top_int)
12	$\neg$(DW_EL_at_bottom$\wedge$ string_at_top_int)
13	$\neg$(EL_release $\wedge$ slips_off_)

9.3.3 Transition properties

State properties answer which properties should be satisfied by the reachable states. Transition properties address issues related to state changes in order to force execution determinism. For example, consider two markings m_1 and m_2. In m_1 the places slips_off_ and EL_release are set to 1 and 0 respectively, which means that the elevator is holding the drill-string and that the slips are not activated. In m_2 the variables slips_off_ and EL-release are set to 0 and 1, respectively, i.e. the power-slips are activated and the elevator is released. Both markings m_1 and m_2 satisfy rule 13 in Table 9.3. However, moving from m_1 to m_2 must guarantee that power-slips are activated before the elevator is released.

One way to do that is to ensure that only one control variable is set at every state change. The places in the pipe handling model represent two types of variables: internal and control variables. We need to make sure that no transition in the reachability graph sets more than one control variable.

9.3.4 Transition Labelling

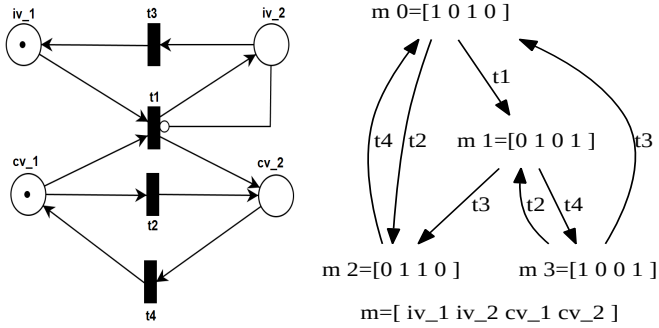
So far we have focus on providing a Petri net model that captures the legal system behaviours for Pipe handling.

A straightforward way to design a control panel is to consider the transitions as control components (buttons). However, this is not always true, since different transitions could set the same variables. This means that it is not necessarily a one-to-one relation between a control button and a Petri net transition. For example, both of the transitions EL_hold_at_top and EL_hold_at_bottom remove one token from EL_release and set one token in EL_hold. The difference is in the condition enabling these transitions, this is why they are modelled as two explicit transitions. However both transitions could be linked to the same control button. In this case, the labels correspond to the control buttons. We say that two transitions share the same label if the following holds:

- They share the same non internal output place (control variable).
- They are never enabled together.

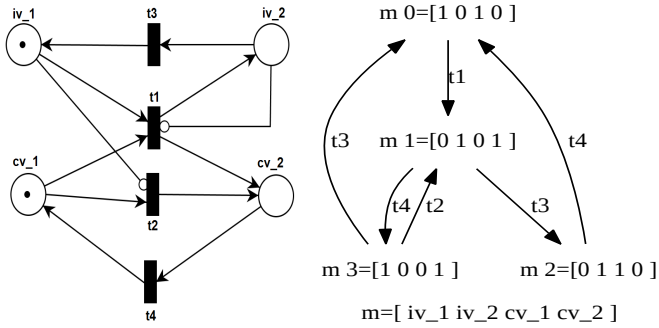
The first condition is to make sure that they control the same variables, while the second condition avoids ambiguities. Avoiding ambiguities means that two transitions that can fire at the same marking can not share the same label. Because from a given marking, choosing a label gives no way to determine which Petri net transition should fire.

Figure 9.8 shows a Petri net and its corresponding reachability graph. The place named cv stand for control variables, while iv stand for internal variables. The figure shows that t_1 and t_2 satisfy the first condition, because they deposit tokens in the same control variables. But the second condition is not satisfied between t_1 and t_2, because there exists a marking m_0 in which they are both enabled.



(a) (b)

Figure 9.8: Ambiguity between t_1 and t_2, they can not share the same label



(a) (b)

Figure 9.9: No ambiguity t_1 and t_2, they can share the same label

The transitions t_1 and t_2 can thus not share the same label. On the other hand, Figure 9.9 shows that there is no ambiguity between t_1 and t_2 as they are never enabled together.

Applying this reasoning on the pipe handling model gives the minimum set of labels that are necessary to control the system dynamics. Finally, the minimum set of labels gives the set of control buttons that the control panel should contain. Table 9.4 summarises the transitions and their corresponding labels.

Table 9.4: Mapping transitions to labels (Control components)

Transitions	**Labels**	**Places**
RA_hold_at_rack, RA_hold_wc	RA_grip	RA_hold
RA_move_pipe_to_rack, RA_to_rack	RA_rack	RA_at_rack
RA_move_pipe_to_wc, RA_to_wc	RA_well_center	RA_at_wc
RA_release_at_rack, RA_release_at_wc	RA_release	RA_release
DW_string_to_bottom, DW_EL_to_bottom	DW_bottom	DW_to_bottom
DW_EL_string_to_top, DW_EL_to_top	DW_top	DW_top
EL_hold_new_pipe, EL_hold_at_bottom, EL_hold_at_top	EL_hold	EL_hold
EL_release_pipe, EL_release_at_bottom, EL_release_at_top	EL_release	EL_release
RN_break_off	break_off	RN_break_off
RN_makeup	RN_makeup	RN_makeup
RN_done_break_off, RN_done_makeup	RN_to_idle	RN_idle
slips_off	slips_off	slips_off_
slips_on	slips_on	slips_on_

9.4 Modelling the Operational Mode

In this section, we present the system model used to handle the actual drilling, which we call the operational mode. In this mode, the machine control imposes a significant influence on the well state. We therefore need to operate the rig machinery with a higher precision. In contrast to pipe handling, the operational model uses control variables that can take large values. This means that the system cannot be

modelled using one-safe Petri net, because not all the control variables are of boolean types.

9.4.1 Using MICPTI nets

In Chapter 6 we proposed MICPTI nets, because P/T nets often fail to capture relatively simple situations. Consider the top-drive, which will rotate the drill-string only if the power-slips are not activated, which in turn can only be activated if the drill-string is not rotating. As simple as it may seem, this situation can not be modelled using P/T nets. On the other hand, MICPTI nets capture quite elegantly the above mentioned case, as shown in Figure 9.10.

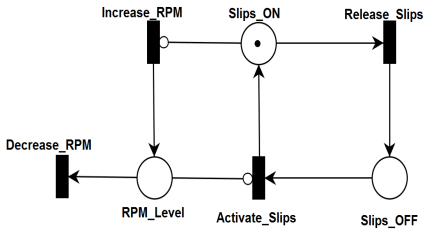


Figure 9.10: Top-drive and power slips modelled using MICPTI

9.4.2 The Operational Model

The actions that a driller can possibly take are modelled as transitions, while the control variables are modelled as places. Table 9.5 summarises the different places and transitions associated to different tools.

The net in Figure 9.11 models the relations between places and transitions for each individual tool, regardless of the interaction between the different devices. We can clearly see that we are dealing with two types of model structures: flat elementary structure and circular elementary structure as defined in Section 6.4. The model in Figure 9.12 relates the different structures according to the MICPTI definition from Section 6.8.

Table 9.5: Places and transitions for the operational mode

Tools	**Places**	**Transitions**
Top-drive	RPM, TD_connect, TD_disconnect	TD_increase_RPM, TD_decrease_RPM, TD_connect, TD_disconnect
Draw-works	DW_downwards_speed, DW_upwards_speed	DW_increase_upwards_speed, DW_decrease_upwards_speed, DW_increase_downwards_speed, DW_decrease_downwards_speed
Power-slips	slips_on_, slips_off_	slips_on, slips_off
Internal BOP	close_ibop, open_ibop	open_ibop, close_ibop
Mud Pump	Flowrate	Pump_increase_Flow, Pump_decrease_Flow

9.5 Operational Model Analysis

9.5.1 General Properties

The fact that the model of Figure 9.12 uses places that can contain large values makes the state space extremely large. Thus, analysing the model by means of its reachability graph is not a feasible approach. In order to analyse the model we assimilate it to an infinite system, allowing places to be unbounded; they can contain an infinite number of tokens.

Fortunately, in Chapter 6 we have developed the necessary tools to analyse a MICPTI model. The general properties of our model are summarised in Table 9.6. The properties are determined using the coverability and characteristic graphs as explained in Section 6.10.

9.5.2 State Properties

As mentioned earlier, the general properties are necessary, but not sufficient conditions for the correct behaviour of the system. Table 9.7 describes the additional requirements that the reachable states must satisfy. Many of the rules are somehow explicitly satisfied by model

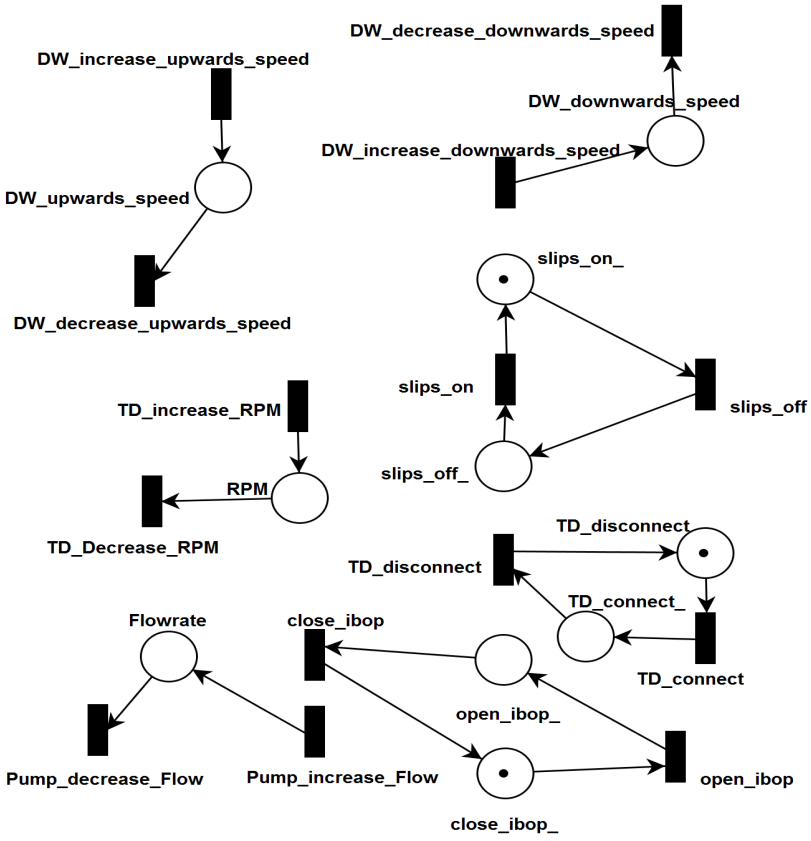


Figure 9.11: Operational model for each tool

construction, but a systematic way of checking that the model satisfies the requirements remains necessary. This is because some rules are modelled explicitly, such as the switching between slips_on_ and slips_off_, where a token is taken from one place and deposited in another. We can therefore say that rule 1 is satisfied by construction.

On the other hand, there are some rules that are not explicitly modelled, but still are satisfied. For example, Rule 4 states that we cannot have a flowrate while the top-drive is disconnected. Rule 4 is not explicitly modelled since there is no inhibitor arc from the place TD_disconnected to the transition increase_flow_rate, but model checking reveals that the rule is satisfied.

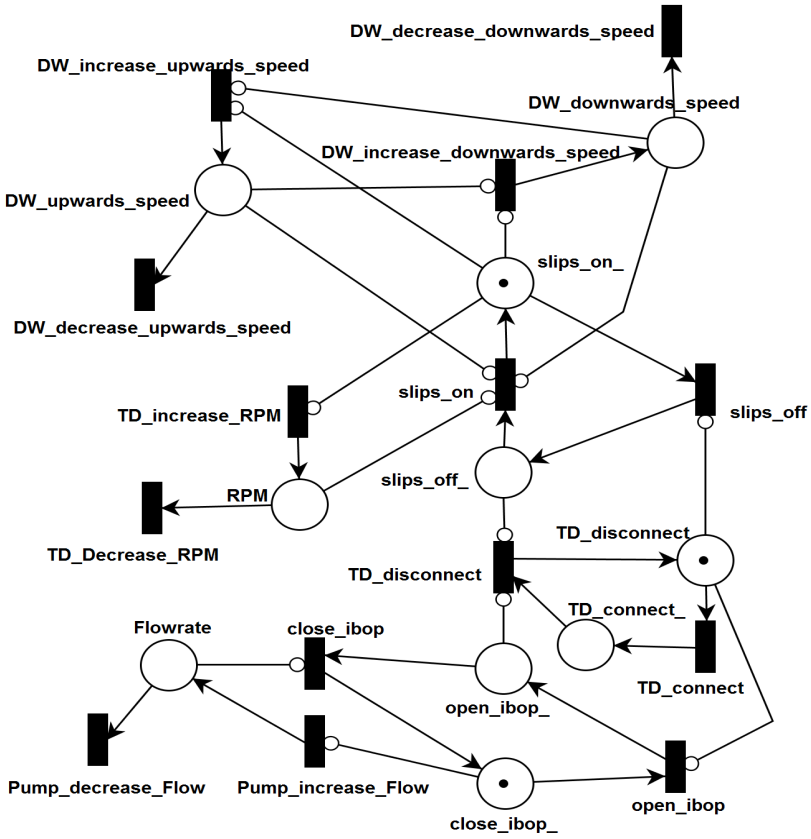


Figure 9.12: Operational model relating the sub-models

9.5.3 Transition properties

These properties are explained in Section 9.3.3, and aim at forcing a certain execution determinism. MICPTI nets guarantee by construction that no more than one set point can be generated by a transition. It is the case because a transition in fes can only set or remove a token from one place. While a transition in *ces* removes a token from one place and deposits a token in another.

Table 9.6: General properties of the operational model

Property	**Description**
MICPTI class	Yes
Nodes	22
Extended markings	17
Unbounded places	DW_downwards_speed, RPM, DW_upwards_speed, Flowrate,
Deadlock states	None
Dead transitions	None
Reversible	Yes
Live	Yes

Table 9.7: State specific rules

Rule	**Formula**
1	slips_on_ $\neq$ slips_off_
2	TD_connect $\neq$ TD_disconnect
3	open_ibop $\neq$ close_ibop
4	$\neg$(Flowrate $> 0 \wedge$ TD_disconnect > 0)
5	$\neg$(Flowrate $> 0 \wedge$ close_ibop > 0)
6	$\neg$(RPM$> 0 \wedge$ slips_on_ > 0)
7	$\neg$(RPM $> 0 \wedge$ TD_disconnect> 0)
8	$\neg$(DW_upwards_speed $> 0 \wedge$ DW_downwards_speed > 0)
9	$\neg$(DW_upwards_speed $> 0 \wedge$ slips_on_ > 0)
10	$\neg$(DW_downwards_speed $> 0 \wedge$ slips_on_ > 0)

9.5.4 Transition Labelling

In Section 9.3.4 we mentioned that two transitions share the same label if the following hold:

- They share the same non internal output place (control variable).
- They are never enabled together.

The model of Figure 9.12 contains no internal variables, that is all the places correspond to control variables. The fact that in a MICPTI

net no two transitions can deposit tokens in the same place makes it obvious that no two transitions can share the same label. Thus, the operational model is such that there is a one-to-one mapping between a transition and its label (control button).

9.6 Assisted Control

In previous sections we divided the system into two subsystems: A pipe handling system and an operational system, where each subsystem is captured in a dedicated Petri net model.

When operating the system in the operational mode the driller may need to switch to the pipe handling mode and vice versa, we therefore need to model the transition between these two modes. We do that by introducing the variable *mode* such that:

- If $mode = 0$ the operational model of Figure 9.12 is the active one.
- If $mode = 1$ the pipe handling model model of Figure 9.2 is the active one.

The Figures 9.13 and 9.14 show a prototype implementation of the drilling control panel under the operational mode and the pipe handling mode respectively. The push buttons on the control panel are mapped to the transitions of the Petri net models as explained in Sections 9.3.4 and 9.5.4. This means that the enabling of a push button is directly related to the enabling of its corresponding Petri net transition. Likewise, pushing a button implies firing its corresponding transition, which in turn leads to a new marking that enables another set of transitions, and thus enable another set of buttons. As Figure 9.13 shows, the initial state of the operational mode ($mode = 0$) is such that it is only possible to connect the top-drive or change to the pipe handling mode ($mode = 1$). The initial state of the pipe handling mode is such that, the driller can choose to move the elevator to the bottom position, cause the rack arm to grip a pipe from the rack, move the rack arm to the well center, or simply move back to the operational mode (see Figure 9.14).

A marking on the other hand is mapped to commands that are further sent to the devices via the command interface. In order to

map markings to commands we need to give interpretations of the tokens. The mapping between command variables, places and their respective token interpretation are summarized in Table 9.8.

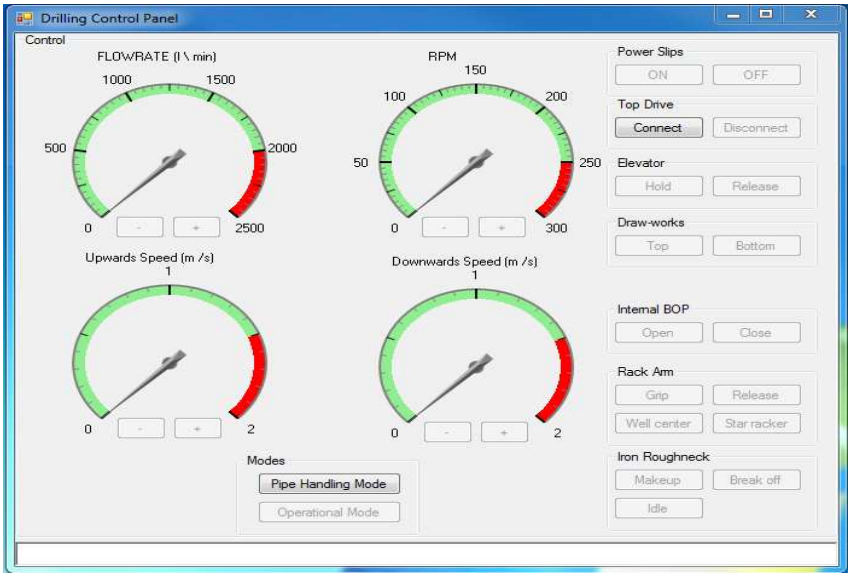


Figure 9.13: Control panel in the initial state of the operational mode

9.6.1 Assisting Processes

We use the approach that was presented in Chapter 7 to model a set of reactive processes. In Chapter 7 we suggested that a process has a triggering condition, exiting condition, and a goal to obtain when it triggers. To show how our approach can be applied to obtain an assisted control of drilling operations, we shall define the following reactive processes:

- Anti-Collision upwards ($proc_1$): This process will trigger to prevent the hook from hitting the Crow-block. It observes the hook position and the hook velocity. When $proc_1$ triggers it aims at obtaining a goal g_1 which in turn is specified by setting the variable UPSPEED=0.
- Anti-Collision downwards($proc_2$): This process will trigger to prevent the hook from hitting the drill-floor. It observes the

Figure 9.14: Control panel in the initial state of the pipe handling mode

hook position and the hook velocity, and triggers by setting the variable DOWNSPEED=0, which is specified by g_2.

- Drill-bit reach bottom ($proc_3$): This process will trigger to prevent the drill-bit from hitting the bottom-hole. It observes the bit depth value, the hook velocity, the rotation, and the flow-rate and triggers by setting the variable DOWNSPEED=0, specified by g_3.
- Move to connection position $proc_4$: This process triggers by the driller when he/she pushes the pipe handling button of Figure 9.13. Upon triggering $proc_4$ sets the variable DOWNSPEED=0.5 m/s (g_4) until the hook position is within a distance of 0 to 1 meters from the drill floor or that the bit-depth is within a distance of 0 to 1 from the total depth. This distance is sufficient to switch to pipe handling mode.
- Set connection mode ($proc_5$): This process triggers under $mode = 0$. Upon triggering it sets the initial conditions of the pipe handling mode (g_5) and sets $mode = 1$.
- High standpipe pressure (pressure in the drill-string) $proc_6$:

Table 9.8: Commands and sensors

Command	**Place**	**Interpretation**
FLOWRATE	Flowrate	10 l/min
UPSPEED	DW_upwards_speed	0.01 m/s
DOWNSPEED	DW_downwards_speed	0.01 m/s
RPM	RPM	1 rotation/ min
SLIPS	slips_on_	ON/OFF
TDCONNECTED	TD_connect	1=ON / 0=OFF
ELHOLD	EL_hold	1=ON/ 0=OFF
DWTOP	DW_EL_at_top	1=ON / 0=OFF
IBOP	open_ibop	1=ON/ 0=OFF
RAGRIP	RA_hold	1=ON/ 0=OFF
RAWC	RA_at_wc	1=ON/ 0=OFF
RNMAKEUP	RN_makeup	1=ON / 0=OFF
RNBREAKOFF	RN_break_off	1=ON / 0=OFF
RNIDLE	RN_idle	1=ON/ 0=OFF

Sensors	**Units**	**Comments**
hpos	m	hook position
td	m	total depth
bd	m	bit depth
torque	m.KN	Torque
SPP	bar	Pressure in the drill-string
HKL	tons	hook load

This process triggers under $mode = 0$. Upon triggering $proc_6$ sets the flow-rate to zero, which is specified by g_6.

- High Torque ($proc_7$): This process triggers under $mode = 0$. Upon triggering $proc_7$ sets the rotation to zero, specified by g_7.
- Connect a new pipe $proc_8$: This process triggers under $mode = 1$. Upon triggering $proc_8$ realizes a pipe connection (g_8), which is obtained by finding a command in which the variable RNMAKEUP=1.
- Move to operational mode ($proc_9$): This process operates in the pipe handling mode ($mode = 1$), and is triggered by the

driller, when he/she pushes the operational mode button. Upon triggering it has to advance the pipe handling system to a state in which it is possible to switch to the operational mode (g_9). The goals g_9 is specified as the initial marking of the pipe handling Petri net model.

The information about the processes of concern is summarised in Table 9.9. The processes that have a priority level 2 can be seen as safety processes, while those with priority 1 can be seen as non critical and are usually triggered by the user. Possible conflicts between different processes are studied using their respective goals. For example, the processes $proc_2$ and $proc_4$ can not run in parallel because their respective goals are mutually exclusive. This exclusion is due to the fact that $proc_2$ when triggered it will reduce the downward speed to zero, while the $proc_4$ will set it to 0.5 m/s.

On the other hand the processes $proc_2$ and $proc_3$ are mutually inclusive, because both of their respective goals (g_2 and g_3) set the downwards speed to zero. The difference between $proc_2$ and $proc_3$ is in their triggering conditions, as the first one prevents the hook collision with drill-floor, while the second prevents the bit collision with the rock (bottom-hole), however their reaction upon triggering remains the same.

The idea is that by combining these processes the driller can be assisted when operating the rig. Typically, if he/she pulls the drill-string such that the hook reaches a position that is above 31 meters, $proc_1$ triggers and immediately sets the upward speed to zero. When the driller wishes to switch from the operational mode to the pipe handling mode, it is sufficient to trigger $proc_4$ which will lower the hook until the appropriate hight above the drill-floor. When the hook reaches a hight that is less than 1 meter above the drill-floor $proc_2$ will trigger, after which $proc_5$ can trigger an cause the mode change. However if the drill-bit is close enough to the bottom hole, $proc_3$ triggers to prevent bit collision with the rock.

9.6.2 Simulation results

In this simulation we consider that the drill-bit is at the depth of 2300 meters, the total depth of the well is 2400 meters, and the hook

Table 9.9: Processes and goals in assisted control

Proc	**Goal**	**Triggering.**	**Exit**	**Mo.**	**Prior**
$proc_1$	g_1	$t_1 = hpos > 31 \wedge UPSPEED \neq 0$	t_1 is false	0	2
$proc_2$	g_2	$t_2 = hpos < 1 \wedge DOWNSPEED \neq 0$	t_2 is false	0	2
$proc_3$	g_3	$t_3 = td - bd < 0.5 \wedge DOWNSPEED \neq 0 \wedge (FLOWRATE = 0 \vee RPM = 0)$	t_3 is false	0	2
$proc_4$	g_4	$t_4 = pos > 1 \wedge (td - bd) > 1$	t_4 is false	0	1
$proc_5$	g_5	$t_5 = hpos < 1$	t_5 is false	0	1
$proc_6$	g_6	$t_6 = SPP > 210$	t_6 is false	0	2
$proc_7$	g_7	$t_7 = torque > 20$	t_7 is false	0	2
$proc_8$	g_8	$t_8 = hpos < 30 \wedge RNMAKEUP = 0$	t_8 is false	1	1
$proc_9$	g_9	$t_9 = hpos > 30 \wedge RNMAKEUP = 1$	t_9 is false	1	1

Goals in operational mode

Goal	**TINC**	**PINC**	**MINC**	**MEX**
g_1		$g_2, g_3, g_4, g_5, g_6, g_7$		
g_2	g_3	g_1, g_5, g_6, g_7	g_3	g_4
g_3	g_2	g_1, g_5, g_6, g_7	g_2	g_4
g_4	g_1	g_6, g_7		g_2, g_3, g_5
g_5	g_1, g_2, g_3, g_6, g_7			g_4
g_6	$g_1, g_2, g_3, g_4, g_5, g_7$			
g_7	$g_1, g_2, g_3, g_4, g_5, g_6$			

Goals in pipe handling mode

Goal	**TINC**	**PINC**	**MINC**	**MEX**
g_8				g_9
g_9				g_8

position is at 30 meters above the drill-floor. The driller is required

to lower the drill-bit until it reaches the bottom of the well. In order to do that, the driller has to connect three stands, as each stand has a length of 30 meters.

Once the drill-bit reaches the bottom hole, the driller is required to start the rotation, and the pump. This is usually done before drilling to obtain a smooth start and avoid too high pressure pulses. After starting the pump and the rotation, the driller increases these two parameters to some predefined values, we use a minimum of 1500 m/l for the flow-rate and 150 rpm for the string rotation, combined with a downwards speed (0.3 to 0.4 m/s). Figure 9.15 and Figure 9.16 show the state evolution under the operational mode and the pipe handling mode respectively.

Steps 0 to 1300

In this interval we can see that the mode has changes three times, corresponding to three stand connections. Prior to each mode change we can see that the process $proc_4$ was triggered. Recall that $proc_4$ sets the speed down to reach the appropriate position for a connection, which is followed by $proc_2$ to prevent the hook from hitting the drill-floor. The process $proc_5$ triggers to set the initial conditions that are necessary before to the pipe handling mode.

When the mode is changed to 1 (pipe handling mode), the process $proc_8$ triggers followed by $proc_9$. The first is responsible to find a sequence of actions that leads to a stand connection, while the latter is responsible to set the initial conditions for changing to the operational mode, this time from 1 to 0. The above mentioned processes are repeated until the bit-depth has reached the total depth.

Steps 1300 to 2400

In this interval the operations are handled manually by the driller. First the rotation and the flow-rate are set to 60 rpm and 400 l/min and further increased to 150 rpm and 1500 l/min with a downward speed of 0.3 m/s. This stage corresponds to the actual drill, because the bit-depth and the total depth increase until the hook reaches the drill-floor, after which $proc_2$ triggers to prevent a collision.

Steps 2400 to 3500

After a stand have been drilled, it is often desired to smooth the well walls, by applying a reduced rotation and flow-rate while moving the drill string; first upwards and then downwards to perform a back-reaming and reaming respectively. When the driller reduces those two parameters, a sudden increase of the stand pipe pressure (SPP) and the torque happens at about step 2600. These events cause $proc_6$ and $proc_7$ to trigger, and set both the flow-rate and the rotation to zero.

The driller then proceeds the operations by doing a back-reaming, when the hook reaches a hight above 30 meters $proc_1$ triggers to prevent a collision with the crow block. The driller then triggers $proc_4$ causing the drill-string to be lowered to the appropriate connection position, followed with a pipe connection as explained earlier.

9.7 Autopilot

In assisted drilling, a driller remains the responsible for the right execution of a drilling program. Talking about auto piloting a rig implies an automated execution of a drilling program. So in addition to supporting the driller with safety processes, we question our selves on whether it is possible to encode a drilling program and run it for the drilling.

We proceed by classifying the processes involved in two categories: planned processes and unplanned processes.

9.7.1 Planned processes

A planned process has as any other process explained so far, a triggering condition and an exiting condition. When the triggering condition is satisfied the process triggers causing the system to reach a state that satisfy it. When the exiting condition is satisfy the process in question signals that its task is finished. The set of planned processes is described in Table 9.10. As the Table shows, we have added the following processes to the ones defined in the assisted drilling section:

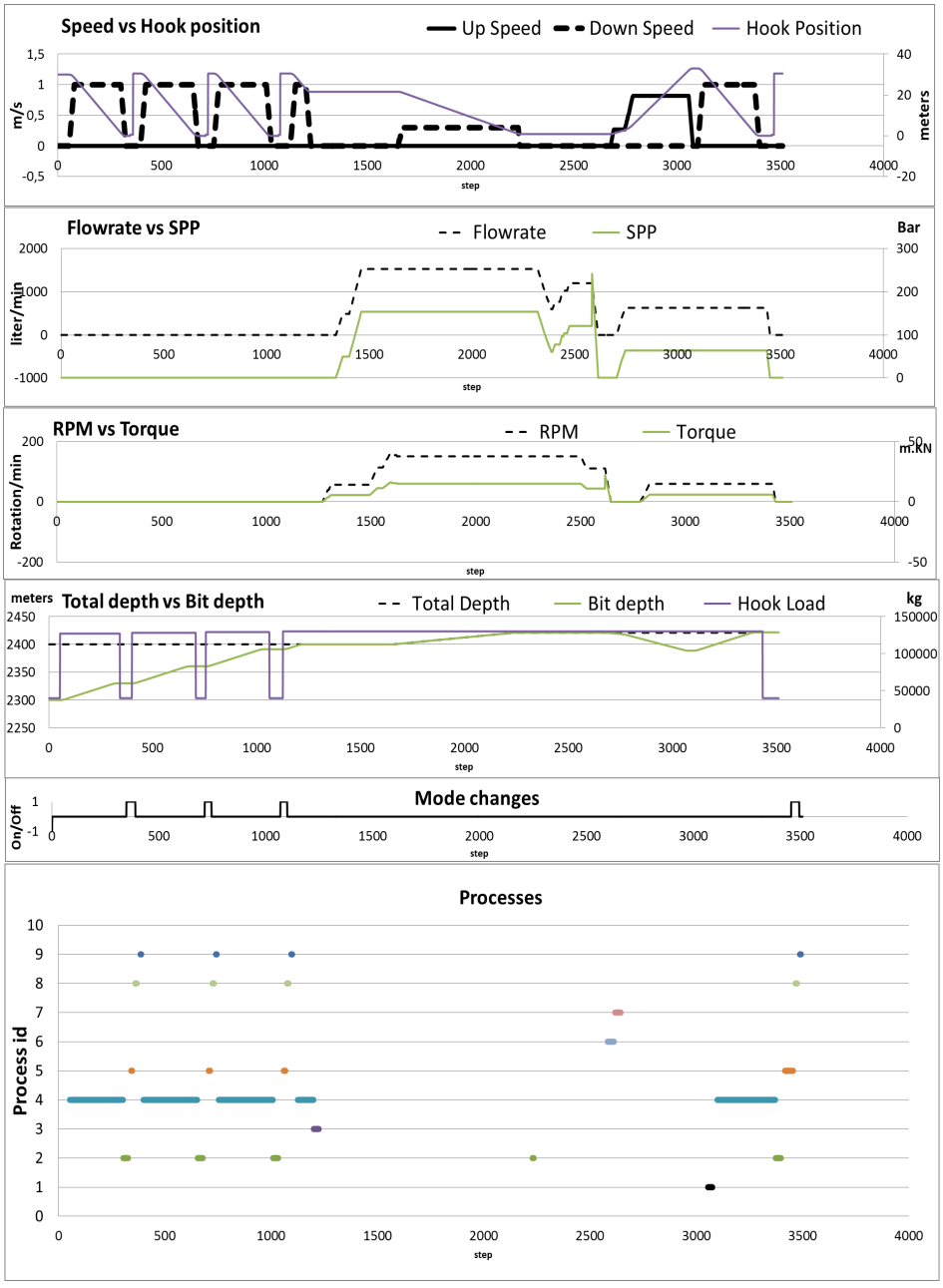


Figure 9.15: Process triggers in assisted drilling scenario

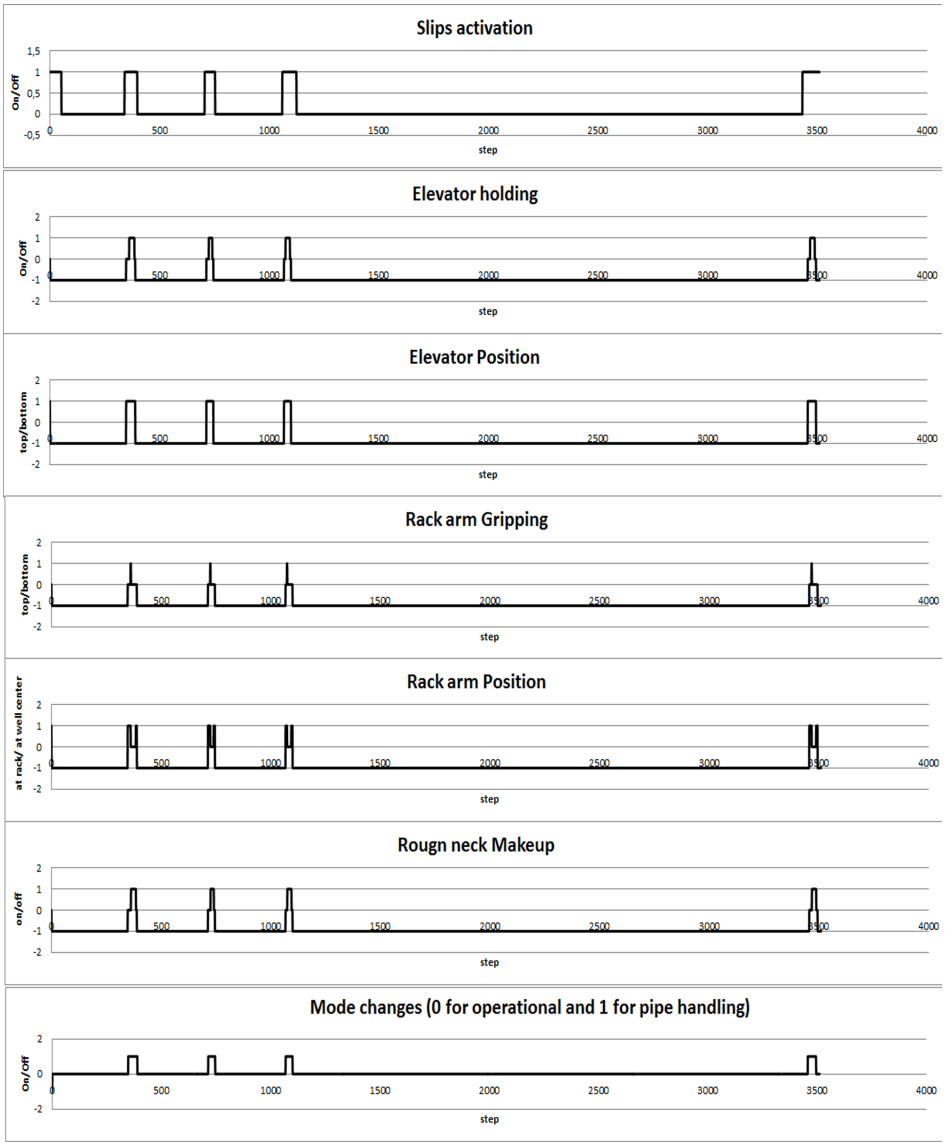


Figure 9.16: State evolution in pipe handling mode

– Initiate Drilling ($proc_{10}$): This process when triggered it will initiate the drilling parameters, by starting the rotation and the mud flow-rate to some predefined set points specified by the goal g_{10}. In our case g_{10} sets the FLOWRATE=400 l/m

and the rotation RPM=60. This process is sometimes called gel-breaking or pump start up and aims at giving a smooth start to drilling.

- Drilling ($proc_{11}$): This process describes the actual drilling, by setting the rotation, the mud flow-rate and the drill-string velocity to some predefined set points specified by the goal g_{11}. This goal sets the FLOWRATE=1800 l/m, the rotation to RPM=150, and DOWNSPEED=0.5 m/s. Normally the drill-string speed is much lower, but we have chosen to use 0.5 m/s to accelerate the experiment somehow and thus avoid a too large data set.
- Back-reaming ($proc_{12}$): This process maintains a certain flow-rate, rpm and an upward velocity. The point is to smooth the well wall after having drilled a stand. Its goal is specified by g_{12}, which sets the FLOWRATE=1000 l/m, the rotation to RPM=60, and UPSPEED=0.5 m/s. We apply a back-reaming of the last 10 meters only.
- Reaming ($proc_{13}$): This process is very similar to $proc_{12}$, the only difference is that it applies to the downward movement expressed by g_{13} in which the variable DOWNSPEED=0.5 m/s.

We say that a process can execute when its triggering conditions are satisfied and it preceding process has finished executing or skipped execution. For example, in order to initiate the pipe handling mode ($proc_5$), we have to make sure that the hook is a the connection level ($proc_4$). If the hook is already at the connection level, the process $proc_4$ needs not to execute and is skipped. So a planned process can either trigger, exit or skip execution.

When exiting or skipping a planned process is systematically enqueued (becomes the last one in queue). More precisely a process skips execution if its turn has come to execute, but its triggering condition is not satisfied. For example if it is the turn of $proc_5$ to execute but the condition t_5 is not satisfied ($hpos < 1$) then $proc_5$ skips executing.

The idea is to organize these processes in such a way that the overall program is satisfactory. This means that a process should be preceded and followed by an appropriate process, and that each one has satisfactory entering, and exiting conditions.

Table 9.10: Planned processes definition

Proc	Goal	Trigger	Exit	Mo.	Prior
$proc_4$	g_4	$t_4 = hpos > 1 \wedge (td - bd) > 1$	t_4 is false	0	1
$proc_5$	g_5	$t_5 = hpos < 1$	t_5 is false	0	1
$proc_8$	g_8	$t_8 = hpos < 30 \wedge RNMAKEUP = 0$	t_8 is false	1	1
$proc_9$	g_9	$t_9 = hpos > 30 \wedge RNMAKEUP = 1$	t_9 is false	1	1
$proc_{10}$	g_{10}	$t_{10} = (hpos + bd) > td \wedge (FLOWRATE \neq 400 \vee RPM \neq 60)$	t_{10} is false	0	1
$proc_{11}$	g_{11}	$t_{11} = (hpos + bd) > td \wedge hpos > 1$	t_{11} is false	0	1
$proc_{12}$	g_{12}	$t_{12} = hpos < 10$	t_{12} is false	0	1
$proc_{13}$	g_{13}	$t_{13} = hpos > 1$	t_{13} is false	0	1

9.7.2 Unplanned processes

If planned process describe what the driller wants to do, unplanned processes describe what he/she wants to avoid. When executing a drilling program, unexpected events may happen, and the program should be interrupted for the time required to cope with the problem.

We say that an unplanned process my trigger at any time during the drilling program execution. Unplanned processes have higher priorities than the planned ones, so that their triggering suspends the execution of a planned process if necessary. These processes are summarized in Table 9.11

9.7.3 Drilling Program Scenario

Now that we have defined the required processes we will illustrate how a basic drilling program could be written. In this scenario we consider that the hook position is 10 meters above the drill floor, the bit-depth is at 2300 meters, while the total depth is at 2305 meters. This means that the drill-string can be lower 5 meters so the bit reaches

Table 9.11: Unplanned processes definition

Proc	Goal	Trigger	Exit	Mo.	Prior
$proc_1$	g_1	$t_1 = hpos > 31 \wedge UPSPEED \neq 0$	t_1 is false	0	2
$proc_2$	g_2	$t_2 = hpos < 1 \wedge DOWNSPEED \neq 0$	t_2 is false	0	2
$proc_3$	g_3	$t_3 = td - bd < 0.5 \wedge DOWNSPEED \neq 0 \wedge (FLOWRATE = 0 \vee RPM = 0)$	t_3 is false	0	2
$proc_6$	g_6	$t_6 = SPP > 210$	t_6 is false	0	2
$proc_7$	g_7	$t_7 = torque > 20$	t_7 is false	0	2

the bottom hole, and that 5 other meters can be drilled, before a new stand can be connected.

A typical part of a drilling program would be to drill a stand, do back-reaming, reaming, go to pipe handling mode, connect a new stand, go back to the operational mode, and repeat the procedure. However, if an exception happens, it should be handled. A drilling program is thus a set of processes set in sequence combined with a set of unplanned processes that can trigger at any time. The planned sequence of planned processes is expressed as follows:

1. $proc_4$ (Go to connection position)
2. $proc_5$ (Go to pipe handling mode)
3. $proc_8$ (Connect a new stand)
4. $proc_9$ (Go to the operational mode)
5. $proc_{10}$ (Initialise drilling)
6. $proc_{11}$ (Drill)
7. $proc_{12}$ (Back-ream)
8. $proc_{13}$ (Ream)
9. Repeat

9.7.4 Simulation Results

We simulate the above described scenario and the results are shown in Figure 9.17 and Figure 9.18 for the operational mode and the pipe handling mode respectively.

Steps 0 to 3000

The first process that triggers is the planned process $proc_4$ requesting a pipe connection by setting the downwards speed to 0.5 m/s. After some steps $proc_3$ triggers to avoid collision of the drill-bit with bottom hole. This means that no connection is possible yet. The succeeding planned processes $proc_5$, $proc_8$ and $proc_9$ will not have their triggering conditions satisfied and thus will systematically skip their turn. The next process that have its triggering condition satisfied is $proc_{10}$ (initiate drilling), causing a flow-rate=400 l/min and a rotation of 60 rpm. After initiating the drilling parameters $proc_{11}$ triggers and the flow-rate is increased to 1800 l/min, the rotation to 150 rpm and the downwards speed to 0.5 m/s. After drilling a stand, $proc_2$ triggers to prevent collision between the hook and the drill-floor. The process $proc_{12}$ triggers after to do a back reaming followed by $proc_{13}$ for reaming, and $proc_2$ to avoid collision with the drill-floor.

Steps 3000 to 7000

This interval starts by $proc_5$ to change the mode to pipe handling mode. and is followed by the planned sequence: $proc_8$, $proc_9$, $proc_{10}$ and $proc_{11}$ before the $proc_2$ triggers to avoid a collision with the drill-floor. After that, $proc_{12}$ for back reaming and $proc_{13}$ for reaming trigger, which again is interrupted by $proc_2$.

Steps 7000 to 9000

Because the condition for pipe connection are already satisfied $proc_4$ is skipped and $proc_5$ triggers for changing to connection mode. After that the planned sequence is executed until $proc_{11}$ (drilling). Due to a sudden increase of the torque stand pipe pressure $proc_6$ trigger to

reduce the flow-rate to zero. The process $proc_7$ triggers to stop the rotation because it observes a high torque.

9.8 Chapter Summary

In this chapter we have applied the theory presented in Part II, to model a drilling control system. We have divided the system into two subsystems: A pipe handling and an operational system. The behaviour of each subsystem is captured in a dedicated Petri net model.

The pipe handling system was modelled using 1-safe Petri nets with inhibitory arcs, and its properties studied using the reachability analysis. The operational model was model using MICPTI nets which we defined and studied in Chapter 6.

Model checking analysis indicate that the presented models are correct, because they satisfy both the general and specific system requirements. The general requirements apply to systems in general where properties like deadlock freeness, transition liveness and reversibility are highly desired.

In the specific properties we made sure that each reachable state satisfies predefined requirements expressed in terms of rules (see Tables 9.3 and 9.7). The states properties make sure that the reachable states are legal, while the transition properties are to ensure that state changes happen in a deterministic manner.

Transition labelling is a concept that we used in order to map several transitions to a common label. There can be several transitions that control the same set point, but are enabled under different conditions. From a human machine interaction (HMI) there is no point of duplicating control components (e.g control button) if not necessary. In practice, the transition labels define the control components of the operator's control station.

We also presented how to obtain an assisted control of the rig using reactive processes. We defined nine reactive processes with varying priorities. We simulated a drilling scenario and observed the triggering of those reactive processes. The simulation shows that our approach can be used to assist a driller when operating a rig, by

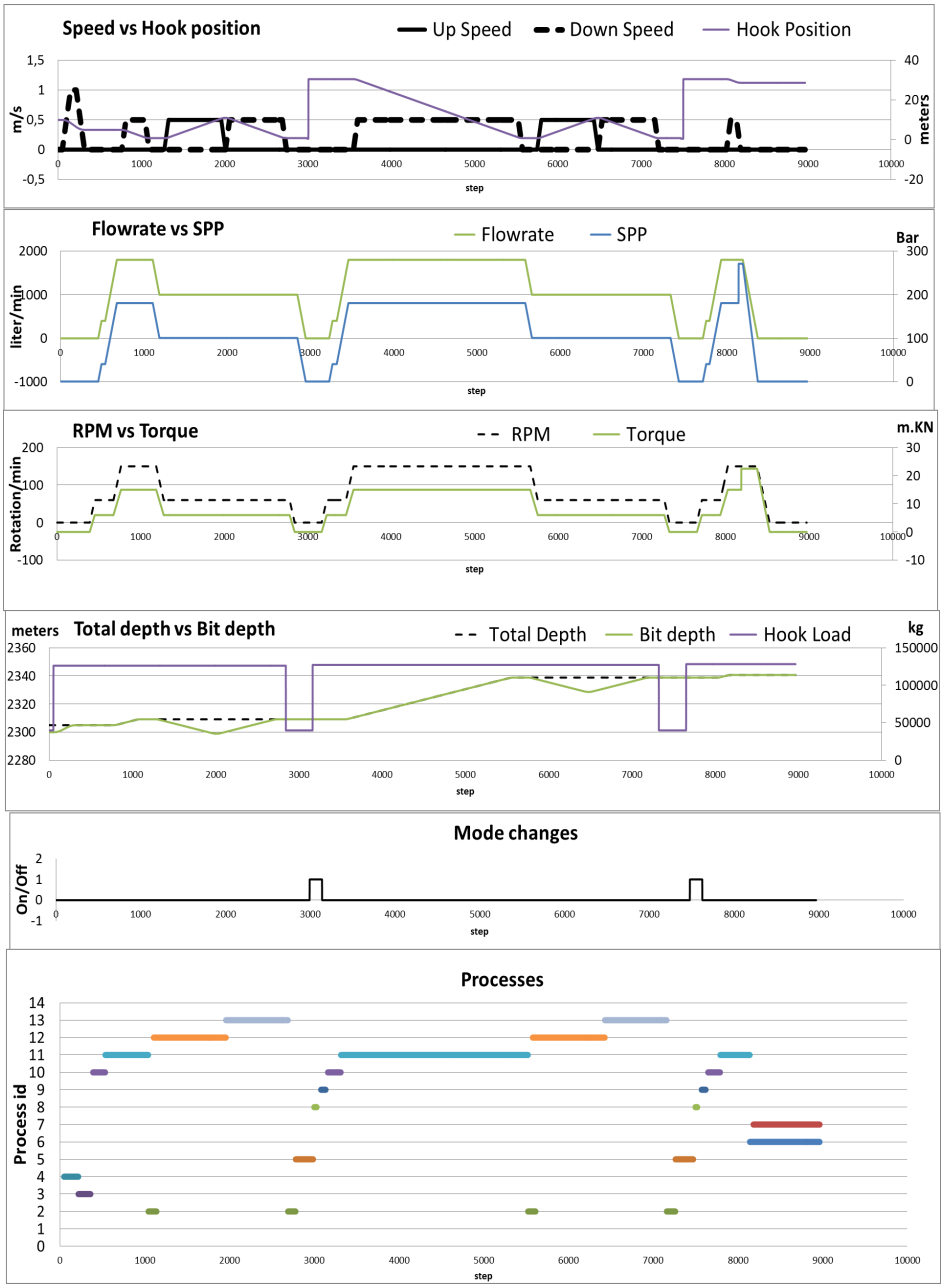


Figure 9.17: Process triggers in automated drilling scenario

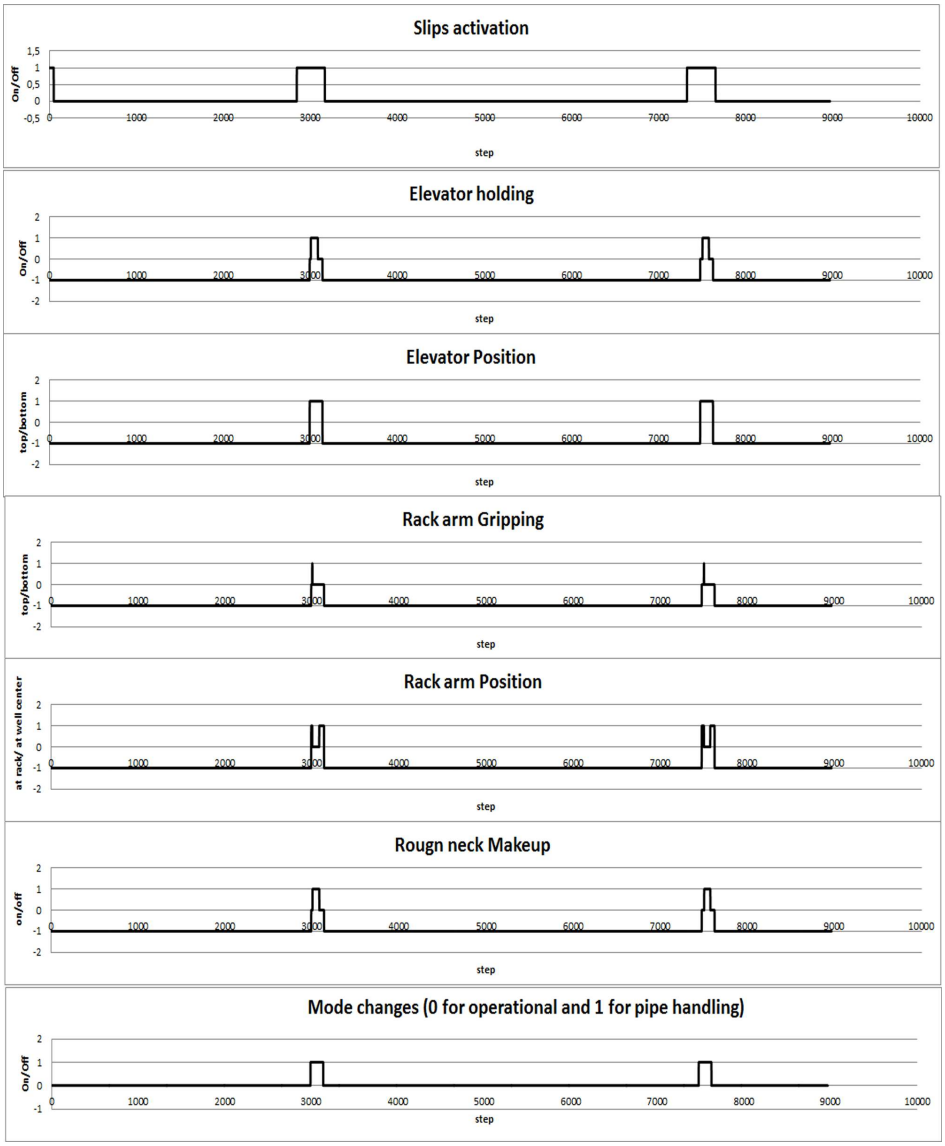


Figure 9.18: State evolution in pipe handling mode

automatically triggering actions for avoiding incidents.

We pushed the assisted control one step a head towards the idea of auto pilot. Using 13 reactive processes we have shown how to represent a plan and run it, even when it could be interrupted.

We divided the reactive processes in two categories: planned and unplanned. Unplanned processes can trigger any time and eventually suspend some ongoing actions or generate others.

Planned processes are organised in a queue, before a processes can trigger its antecedent must have finished running. In addition, a planned process could skip execution when its triggering condition is not satisfied.

We have shown that an appropriate combination of reactive processes could lead to a satisfactory emergent behaviour of the overall system. More precisely, we have managed to represent a tripping sequence, combined with a repetition of a stand connection and drilling even with the presence of incidents.

Part IV

Conclusions

Chapter 10

Conclusions and Further Work

This thesis aimed at proposing an approach for advancing existing drilling control system a step towards the autonomous drilling vision.

Main Contributions

The main contribution of this thesis was a theoretical framework for improving today's drilling control systems. Existing systems lack supervisory control and the ability to handle incidents. Those two limitations are the main motivation of this thesis.

We viewed the drilling control system as a combination of two entities in interaction. From one side, we have a rig which is the tool by which a well is drilled, and drilling problems are handled. From another side, we have a well which dynamics is influenced by the rig, but is not totally controlled by it.

Tightly coupling the rig with the well dynamics is probably the main tendency in the drilling control industry. However, this coupling has a fundamental problem: If we cannot fully control the well dynamics, it will only be more difficult to control the rig dynamics. This idea was presented in Chapter 4 concluding by that a separation of concerns between the well and the rig is necessary if we want to verify the correctness of the system.

Our first problem was to address the control of the rig dynamics. We categorised the rig as a DES and suggested an approach for modelling its behaviour. Because a rig is a critical system we wanted to make sure that its behaviour always satisfies the specifications. In other words, we needed a modelling formalism that has; sufficient modelling, and decision power.

After a literature study we found out that Petri nets could be a good candidate for modelling the rig dynamics, but not without problems. We found it necessary to use P/T nets extended with inhibitor arcs (PTI nets), but this type of arcs cause significant problems, because they reduce drastically the decision power, and thus no system properties could be checked.

We therefore proposed a class of PTI nets that uses inhibitor arcs in a restrictive manner such that it preserves a high decision power. We have first presented the CPTI class, and have shown how to compute its coverability. We have also shown how to determine properties such as deadlock or boundedness. We later on presented a subclass of CPTI called MICPTI nets on which it is possible to determine most of the properties of interst using a combination of the net's coverability and characteristic graphs. Finally we have shown that the MICPTI class elegantly captures parts of the rig dynamics. This class was studied in Chapter 6.

The next problem that we faced is to enable automated responses to well incidents. For that, we introduced the concept of *reactive processes* modelled on top of Petri nets. A *reactive process* observes some key parameters, and triggers when some conditions are satisfied. Once a *reactive process* triggers it aims at achieving a *goal*. In our approach we associated each reactive process' goal with a set of states and used that association to systematically study the interactions between those processes.

Typically, two reactive processes could share a common goal, or have conflicting goals. When the number of processes increases it becomes difficult to understand the interaction between them, and a formal method is required. This has been addressed in Chapter 7.

Finally, we have applied the proposed theory to model a drilling control system and supported it with reactive processes for handling incidents. Furthermore, we have extended the *reactive process*

concept to represent a portion of a drilling plan, and have shown how such a plan could be executed even with the presence of incidents. As a result, we have demonstrated that a careful combination of different *reactive processes* could lead to an autonomous emergent behaviour.

Direction for Future Research

Each of the addressed problems could potentially be improved. This section suggests directions for future work that are relevant for three topics: Petri nets, reactive processes, and drilling control systems.

Petri nets

The class of PTI nets presented in Chapter 6 could probably be enlarged, and eventually defined differently. In our work we found in the Well-Structured-Transition-Systems (WSTS) [47] theory plausible explanations about why PTI nets are fundamentally problematic. Inspired by WSTS, we have managed to restrict the use of inhibitor arcs in such a way that the $T-monotonicity$ and $S-monotonicity$ are satisfied. However, it would be interesting to find other restrictions or even some mother classes for the MICPTI and CPTI nets and eventually enlarge their modelling power.

From a practical side, we have used a Petri net tool called PIPE [18]. Our use of that tool was mainly to draw nets and have a quick understanding of their dynamics. It could be interesting to include our defined classes and their analysis to that tool. Another tool called GPenSIM [33] is less graphical but is more general and could also benefit from such an extension.

Reactive processes

The reactive processes were defined through a triggering condition, an exiting condition and a *goal*. The *goal* could be specified more flexibly using some operators such as $<$, $>$, $\leq$, rather than just $=$.

We have so far studied the interaction between *reactive processes* through the states associated to *goals*. However no knowledge on

the expected effect on the environment has been encoded. Such an encoding could help finding correlations between reactive processes.

Typically, a process $proc_1$ that triggers to lift the drill-string has an immediate effect of increasing the hook position. Another process $proc_2$ that triggers when the hook position has reached a certain height, should in principle always trigger after $proc_1$. That is, triggering $proc_2$ is the consequence of triggering $proc_1$. Identifying those correlations could help a better anticipation of the overall system behaviour.

Drilling Control Systems

There already exists a simulator of the well dynamics [43], where given some inputs the simulator generates synthetic sensor data. However the drilling control domain is missing a simulator for actually improving the control it self.

A good initiative would be to propose an open architecture of drilling control systems. Such an architecture should be agreed on with some industrials in order to provide a realistic simulator. The main benefits of taking this initiative would to bring open drilling control problems closer to researchers, who would focus on the actual problems rather than take their own assumptions,

Furthermore, benchmarks could be defined on specific problems: typically, realizing control supervision, identifying critical situations, determine actions to critical situations, autonomous execution of drilling plans etc. In other words, the work done in this thesis could be concretely compared to others methods, if benchmarks existed.

List of Publications

10.1 Relevant

1. N. Saadallah, Hein Meling, and B. Daireaux. A simple machine in a complex environment: A petri net approach. In *Intelligent Engineering Systems (INES), 2011 15th IEEE International Conference on*, pages 387 –392, june 2011. doi: 10.1109/INES.2011.5954778

2. N. Saadallah, H. Meling, and B. Daireaux. Modeling a drilling control system, as a discrete-event-system. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1 –5, march 2011. doi: 10.1109/CCCA.2011.6031461

3. Nejm Saadallah and Benoit Daireaux. A goal based approach on top of Petri nets. In *International Workshop on Petri Nets and Software Engineering (PNSE2011)* (Poster)

10.2 Less Relevant

1. A. Riid and N. Saadallah. Unsupervised learning of well drilling operations: Fuzzy rule-based approach. In *Intelligent Engineering Systems (INES), 2012 IEEE 16th International Conference on*, pages 375 –380, june 2012. doi: 10.1109/INES.2012.6249862

2. Benoit Daireaux Nejm Saadallah. Drilling control system modelling approach. In *IADC World Drilling 2011 Conference and Exhibition, Copenhagen*, 2011

Bibliography

[1] Book review: Coloured petri nets: Basic concepts, analysis methods and practical use (volume 1) by kurt jensen: (springer-verlag, 1992). *SIGOPS Oper. Syst. Rev.*, 28(1):1–2, January 1994. ISSN 0163-5980. URL `http://dl.acm.org/citation.cfm?id=164853.871713`. Reviewer-Nutt, Gary J.

[2] Van Der Aalst. The application of petri nets to workflow management, 1998.

[3] Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521875463.

[4] Tilak Agerwala and Mike Flynn. Comments on capabilities, limitations and correctness of petri nets. *ACM Sigarch Computer Architecture News*, 2:81–86, 1973. doi: 10.1145/633642.803973.

[5] Alain and Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144 – 179, 1990. ISSN 0890-5401. doi: 10.1016/0890-5401(90)90009-7.

[6] Hassane Alla and Rene David. A modelling and analysis tool for discrete events systems: continuous petri net. *Performance Evaluation*, 33(3):175 – 199, 1998. ISSN 0166-5316.

[7] Charles Andre and Marie-Agnes Peraldi. Grafcet and synchronous languages.

[8] Seabed Rig A/S. URL http://www.seabedrig.com/.

[9] J.J. Azar and G.R. Samuel. *Drilling Engineering*. PennWell Corporation, 2007. ISBN 9781593700720. URL http://books.google.no/books?id=eseViO982VgC.

[10] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, May 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.07.036.

[11] R. Baker and University of Texas at Austin. Petroleum Extension Service. *A Primer of Oilwell Drilling: A Basic Text of Oil and Gas Drilling*. Petroleum Extension Service, Continuing & Extended Education, University of Texas at Austin, 2001. ISBN 9780886981945. URL http://books.google.no/books?id=-LGqQgAACAAJ.

[12] J. Bellingham and T. Consi. State configured layered control. *Proc. of the IARP 1st Workshop on: Mobile Robots for Subsea environments*, page 7580, 1990.

[13] Gerard Berry. Proof, language, and interaction. chapter The foundations of Esterel, pages 425–454. MIT Press, Cambridge, MA, USA, 2000. ISBN 0-262-16188-5.

[14] Gerard Berry and Georges Gonthier. The esterel synchronous programming language: design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, November 1992. ISSN 0167-6423.

[15] Eike Best, Raymond R. Devillers, and Maciej Koutny. Petri nets, process algebras and concurrent programming languages. In *Lectures on Petri Nets II: Applications, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 1–84, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65307-4. URL http://dl.acm.org/citation.cfm?id=647445.727064.

[16] Jonathan Billington, Søren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian

Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *Applications and Theory of Petri Nets 2003: 24th International Conference*, pages 1023–1024, Eindhoven, The Netherlands, June 2003. URL `http://www.springerlink.com/content/rp1dqtlmqr5q665b`.

[17] Erhan Isevcan Bjarne Bennetzen, John Fuller. Extended reach wells. *Oilfield Review*, 22(3), 2010.

[18] Pere Bonet, Catalina Llado, Ramon Puijaner, and William Knottenbelt. Pipe v2.5.: a petri net tool for performance modelling. In *23rd Latin American Conference on Informatics*, October 2007.

[19] Rodney A. Brooks. A robust layered control system for a mobile robot. 1985.

[20] Joanna J. Bryson. Intelligence by design: Principles of modularity and coordination for engineering complex adaptive agents. June 2001. AI Technical Report 2001-003.

[21] Joanna J. Bryson. Action selection and individuation in agent based modelling. pages 317–330, 2003.

[22] H. K. Buning, T. Lettman, and E. W. Mayr. Projections of vector addition system reachability sets are semilinear. 1988.

[23] Nadia Busi. Analysis issues in petri nets with inhibitor arcs. *Theor. Comput. Sci.*, 275:127–177, March 2002. ISSN 0304-3975. doi: 10.1016/S0304-3975(01)00127-X. URL `http://dl.acm.org/citation.cfm?id=570571.570576`.

[24] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: a declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '87, pages 178–188, New York, NY, USA, 1987. ACM. ISBN 0-89791-215-2. doi: 10.1145/41625.41641.

[25] Christos G. Cassandras and Stephane Lafortune. Introduction to discrete event systems. 2006.

[26] Daireaux Benoit Drilling Cayeux Eric and Norway Well Modeling Group, International Research Institute of Stavanger. From machine control to drilling control. In *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*.

[27] Claudine Chaouiya, Elisabeth Remy, and Denis Thieffry. Petri net modelling of biological regulatory networks. *J. of Discrete Algorithms*, 6(2):165–177, June 2008. ISSN 1570-8667.

[28] Creative Commons. Attribution 3.0 unported. URL `http://creativecommons.org/licenses/by/3.0/deed.en`.

[29] Greg Conran. Horizontal and complex-trajectory wells. *The Journal of Petroleum Technology*, November 2009.

[30] R. David and H. Alla. *Discrete, Continuous, And Hybrid Petri Nets*. Springer, 2005. ISBN 9783540224808.

[31] Rene David. Grafcet: a powerful tool for specification of logic controllers.

[32] Rene David and Hassane Alla. *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. ISBN 013327537X.

[33] Reggie Davidrajuh. *Modeling and Simulation of Discrete Event Systems with Petri Nets: A Hands-On Approach with GPenSIM*. VDM Verlag, Saarbrücken, Germany, Germany, 2009. ISBN 3639195663, 9783639195668.

[34] A.A. Desrochers, R.Y. Al-Jaar, and IEEE Control Systems Society. *Applications of petri nets in manufacturing systems: modeling, control, and performance analysis*. IEEE Press, 1995. ISBN 9780879422950.

[35] S. Devereux. *Drilling Technology in Nontechnical Language*. PennWell nontechnical series. PennWell, 1999. ISBN 9780878147625. URL `http://books.google.no/books?id=cYFx8n-1mdMC`.

[36] Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):pp. 413–422, 1913. ISSN 00029327. URL `http://www.jstor.org/stable/2370405`.

[37] S. Donatelli, M. Ribaudo, and J. Hillston. A comparison of performance evaluation process algebra and generalized stochastic petri nets. In *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models*, PNPM '95, pages 158–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7210-2. URL `http://dl.acm.org/citation.cfm?id=826033.826775`.

[38] Catherine Dufourd, Alain Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. pages 103–115, 1998.

[39] Hartmut Ehrig, Wolfgang Reisig, Grzegorz Rozenberg, and Herbert Weber, editors. *Petri Net Technology for Communication-Based Systems - Advances in Petri Nets*, volume 2472 of *Lecture Notes in Computer Science*, 2003. Springer. ISBN 3-540-20538-1.

[40] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz - Open Source Graph Drawing Tools. *Graph Drawing*, pages 483–484, 2001.

[41] Erik Dvergsnes Eric cayeux, Benoit Daireaux. Automation of drawworks and topdrive management to minimize swab/surge and poor-downhole-condition effect. In *IADC/SPE Drillng Conference and Exhibition, New Orleans, Louisiana, USA*, 2-4 February 2010.

[42] Erik Dvergsnes Eric cayeux, Benoit Daireaux. Automation of mud-pump management: Application to drilling operations in the north sea. *SPE Drilling and Completion*, 26, Nr 1:41–51, 2011.

[43] Erik Wolden Dvergsnes Amare Leulseged IRIS Bjrn Torstein Bruun Statoil Mike Herbert ConocoPhillips Eric Cayeux, Benoit Daireaux. Advanced drilling simulation environment for testing new drilling automation techniques. *2012. IADC/SPE Drilling Conference and Exhibition*, 2012.

[44] Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. 1994.

[45] Angela Di Febbraro and Nicola Sacco. On modelling urban transportation networks via hybrid petri nets. *Control Engineering Practice*, 12(10):1225 – 1239, 2004.

[46] Colin Fidge. A comparative introduction to csp, ccs and lotos. Technical report, Software Verification Research Centre Department of Computer Science The University of Queensland Queensland 4072, Australia, 1994.

[47] Alain Finkel. A generalization of the procedure of karp and miller to well structured transition systems. pages 499–508, 1987.

[48] Alain Finkel. The minimal coverability graph for petri nets. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1993*, pages 210–243, London, UK, UK, 1993. Springer-Verlag. ISBN 3-540-56689-9. URL `http://dl.acm.org/citation.cfm?id=647738.735634`.

[49] Alain Finkel, Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. A counter-example the minimal coverability tree algorithm. Technical Report 535, Université Libre de Bruxelles, Belgium, 2005.

[50] E.W. Dvergsnes J.E. Gravdal F.P. Iversen, E. Cayeux, NOV; A. Torsvoll E.H. Vefring, Intl. Research Inst. of Stavanger (IRIS); B. Mykletun, Statoil; S. Omdal, and Eni Agip SpA A. Merlo. Monitoring and control of drilling utilizing continuously updated process models. In *IADC/SPE Drilling Conference, 21-23 February 2006, Miami, Florida, USA*, 2006.

[51] Abdoulaye Gamati. *Designing Embedded Systems with the SIGNAL Programming Language - Synchronous, Reactive Specification*. Springer, 2010. ISBN 978-1-4419-0940-4.

[52] Emden Gansner, Eleftherios Koutsofios, and Stephen North. *Drawing graphs with dot*, January 2006. URL `http://www.graphviz.org/Documentation/dotguide.pdf`.

[53] Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. On the efficient computation of the minimal coverability set for petri nets. In *Proceedings of the 5th international conference on Automated technology for verification and analysis*, ATVA'07, pages 98–113, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75595-0, 978-3-540-75595-1. URL `http://dl.acm.org/citation.cfm?id=1779046.1779057`.

[54] Latefa Ghomri and Hassane Alla. Modeling and analysis using hybrid petri nets. *Nonlinear Analysis: Hybrid Systems*, 1(2): 141 – 153, 2007.

[55] D. Gielen, J. Podkański, International Energy Agency, Organisation for Economic Co-operation, and Development. *Prospects For Co2 Capture And Storage*. Energy Technology Analysis. OECD/IEA, 2004. ISBN 9789264108813. URL `http://books.google.no/books?id=tDg9TtI3xQsC`.

[56] Alessandro Giua. *Petri Nets as Discrete Event Models for Supervisory Control*. PhD thesis, Rensselaer Polytechnic Institute (Troy, New York), 1992.

[57] Ursula Goltz. On representing ccs programs by finite petri nets. In *Proceedings of the Mathematical Foundations of*

Computer Science 1988, MFCS '88, pages 339–350, London, UK, UK, 1988. Springer-Verlag. ISBN 3-540-50110-X. URL `http://dl.acm.org/citation.cfm?id=645718.665801`.

[58] Steven Gordon and Jonathan Billington. Modelling the wap transaction service using coloured petri nets. In *Proceedings of the First International Conference on Mobile Data Access, LNCS 1748*, pages 16–17. Springer-Verlag. ISBN, 1999.

[59] T. G. I. Group. Petri Nets World: Online Services for the International Petri Net Community. Technical report, University of Hamburg, 2007.

[60] Tianlong Gu and Parisa A. Bahri. A survey of petri net applications in batch processes. *Comput. Ind.*, 47(1):99–111, January 2002. ISSN 0166-3615.

[61] M. Hack. Analysis of production schemata by petri nets. Technical report, Cambridge, MA, USA, 1972.

[62] M. Hack. Petri net language. Technical report, Cambridge, MA, USA, 1976.

[63] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 1441951334, 9781441951335.

[64] Simon Hardy and Pierre N. Robillard. Pn: Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. *J Bioinform Comput Biol*, 2004: 2–4, 2004.

[65] D. Harel and A. Pnueli. *On the development of reactive systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-15181-8.

[66] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[67] David Harel. On visual formalisms. *Commun. ACM*, 31(5): 514–530, 1988.

[68] Monika Heiner, David Gilbert, and Robin Donaldson. Petri nets for systems and synthetic biology. In *SFM*, pages 215–264, 2008.

[69] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359585.

[70] L. E. Holloway, B. H. Krogh, and A. Giua. A Survey of Petri Net Methods for Controlled Discrete EventSystems. *Discrete Event Dynamic Systems*, 7(2):151–190, April 1997. ISSN 0924-6703. doi: 10.1023/A:1008271916548. URL `http://dx.doi.org/10.1023/A:1008271916548`.

[71] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003. ISBN 0-321-22862-6.

[72] J. Hong and D. Bae. HOONets: Hierarchical Object-Oriented Petri Nets for System Modeling and Analysis, 1998. URL `http://citeseer.ist.psu.edu/hong98hoonets.html`.

[73] Sigve Hovda, Henrik Wolter, Glenn-Ole Kaasa, and Tor Stein Olberg. Potential of ultra high-speed drill string telemetry in future improvements of the drilling process control. In *IADC/SPE Asia Pacific Drilling Technology Conf. and Exhibition*, 2008.

[74] B. Hrz and M. C. Zhou. *Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 184628872X, 9781846288722.

[75] Marian V. Iordache and Panos J. Antsaklis. *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Birkhauser, 2006. ISBN 0817643575.

[76] Fionn Iversen, Eric Cayeux, Erik Wolden Dvergsnes, Ragna Ervik, Morten Welmer, and Mohsen Karimi Balov. Offshore field test of a new system for model integrated closed-loop drilling control. *SPE Drilling and Completion*, 24:518–530, 2009.

[77] F.P. Iversen, E. Cayeux, E.W. Dvergsnes, J.E. Gravdal, E.H. Vefring, B. Mykletun, A. Torsvoll, S. Omdal, and A. Merlo. Monitoring and control of drilling utilizing continuously updated process models. In *IADC/SPE Drilling Conf.*, 2006.

[78] M. Jantzen. Language theory of petri nets. In *Advances in Petri nets 1986, part I on Petri nets: central models and their properties*, pages 397–412, London, UK, UK, 1987. Springer-Verlag. ISBN 0-387-17905-4. URL `http://dl.acm.org/citation.cfm?id=28641.28655`.

[79] K. Jensen. An introduction to the practical use of coloured petri nets. *Lecture Notes in Computer Science: Lectures on Petri Nets II: Applications*, 1492, 1998.

[80] Kurt Jensen. High level petri nets. In Pagnoni, A. and Rozenberg, G., editors, *Informatik-Fachberichte 66: Application and Theory of Petri Nets — Selected Papers from the Third European Workshop on Application and Theory of Petri Nets, Varenna, Italy, September 27–30, 1982*, pages 166–180. Springer-Verlag.

[81] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and CPN tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 2007.

[82] P. T. Laney K. K. Bloomfield. Estimating well costs for enhanced geothermal system applications. Technical report, Idaho National Laboratory, August 2005.

[83] David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms, 1997.

[84] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 1969.

[85] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, July 1976. ISSN 0001-0782. doi: 10.1145/360248.360251.

[86] Dënes Koenig. *Theory of finite and infinite graphs*. Birkhauser Boston Inc., Cambridge, MA, USA, 1990. ISBN 0-8176-3389-8.

[87] Vedran Kordic, editor. *Petri Net, Theory and Applications*. I-Tech Education and Publishing, 2007.

[88] R.W. Lewis and Institution of Electrical Engineers. *Programming Industrial Control Systems Using Iec 1131-3*. IEE Control Engineering Series. Institution of Electrical Engineers, 1998. ISBN 9780852969502. URL `http://books.google.no/books?id=sc-g9k6dPzMC`.

[89] J.W.Jenner L.J.Ayling and Maris International Ltd J.M.Neffgen. Seabed located drilling rig - itf pioneer project. *Offshore Technology Conference, 5 May-8 May 2003, Houston, Texas*, May 2003.

[90] West Drilling Products AS Helge Krohn West Drilling Products AS Mads Grinrd, SPE. Continuous motion rig. a detailed study of a 750 ton capacity , 3600 m/hr trip speed rig. In *SPE/IADC Drilling Conference and Exhibition, 1-3 March 2011, Amsterdam, The Netherlands*, 2011.

[91] Well System Technology A/S Mads Grinrod, SPE. Continuous motion rig: A step change in drilling equipment. In *IADC/SPE Drilling Conference and Exhibition, 2-4 February 2010, New Orleans, Louisiana, USA*, 2010.

[92] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992. ISBN 0-387-97664-7.

[93] F. Maraninchi and Y. Remond. Argos: an automaton-based synchronous language. *Computer Languages*, (27):61–92, 2001.

[94] Ernst W. Mayr. Persistence of vector replacement systems is decidable. pages 309–318, 1981.

[95] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. ISBN 0387102353.

[96] Robin Milner. Calculi for synchrony and asynchrony * part i . synchrony. *Science*, 25(3):267–310, 1983.

[97] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0-521-65869-1.

[98] Toshiyuki Miyamoto and Sadatoshi Kumagai. A survey of object-oriented petri nets and analysis methods. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E88-A:2964–2971, November 2005. ISSN 0916-8508. doi: http://dx.doi.org/10.1093/ietfec/e88-a.11.2964. URL `http://dx.doi.org/10.1093/ietfec/e88-a.11.2964`.

[99] Tadao Murata. Petri nets: Properties, analysis and applications. pages 541–580, April 1989.

[100] M. Nagasaki, A. Doi, H. Matsuno, and S. Miyano. A versatile petri net based architecture for modeling and simulation of complex biological processes. *Genome informatics. International Conference on Genome Informatics*, 15(1):180–197, 2004.

[101] Benoit Daireaux Nejm Saadallah. Drilling control system modelling approach. In *IADC World Drilling 2011 Conference and Exhibition, Copenhagen*, 2011.

[102] Petroleum Safety Authority Norway. Risk level in norwegian petroleum activities development trends 2011 - the norwegian shelf. Technical Report 47, 2011.

[103] Yale University. Dept. of Computer Science and R.J. Lipton. *The Reachability Problem Requires Exponential Space*. Research report (Yale University. Dept. of Computer Science). Department of Computer Science, Yale University, 1976. URL `http://books.google.no/books?id=7iSbGwAACAAJ`.

[104] Jens Ingvald Ornaes. Closed-loop control for decision-making applications in remote operations. In *IADC/SPE Drilling Conference and Exhibition, 2-4 February 2010, New Orleans, Louisiana, USA*, 2010.

[105] Posc Caesar Association (PCA). "rcn/nfr project :autoconrig. URL `https://www.posccaesar.org/wiki/IOHN/AutoConRig`.

[106] James L. Peterson. A note on colored petri nets. *Information Processing Letters, Vol.11, No.1*, 11:40–43, 1980.

[107] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981. ISBN 0136619835.

[108] Carl Adam Petri. Communication with automata. 1966.

[109] Petroleumstilsynet. Risikoniv i petroleumsvirksomheten hovedrapport, utviklingstrekk 2011, norsk sokkel. Technical Report 261, 2011.

[110] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[111] P. Ramadge and W. Wonham. Supervisory Control of a Class of Discrete Event Processes. *Siam J. Control and Optimization*, 25(1), 1987. URL `http://locus.siam.org/SICON/volume-25/art_0325013.html`.

[112] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.

[113] Venkatramana N. Reddy, Michael N. Liebman, and Michael L. Mavrovouniotis. Qualitative analysis of biochemical reaction systems. *Computers in Biology and Medicine*, 26(1):9 – 24, 1996. ISSN 0010-4825. doi: 10.1016/0010-4825(95)00042-9.

[114] A. Riid and N. Saadallah. Unsupervised learning of well drilling operations: Fuzzy rule-based approach. In *Intelligent Engineering Systems (INES), 2012 IEEE 16th International Conference on*, pages 375 –380, june 2012. doi: 10.1109/INES.2012.6249862.

[115] George W. Halsey Rolv Rommetveit, Knut S. Bjorkevoll, Hitec Products Drilling; Mike Herbert ConocoPhillips; Ove Sandve First Interactive; Erling Fjr, SINTEF Petroleum Research; Sven Inge Odegaard, and Aker Kvaerner Maritime Hydraulics Bjarne Larsen. e-drilling: A system for real-time drilling simulation, 3d visualization and control. *Digital Energy Conference and Exhibition, 11-12 April 2007, Houston, Texas, U.S.A.*, 2007.

[116] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. ISBN 0137903952. URL `http://portal.acm.org/citation.cfm?id=773294`.

[117] N. Saadallah, H. Meling, and B. Daireaux. Modeling a drilling control system, as a discrete-event-system. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1 –5, march 2011. doi: 10.1109/CCCA.2011.6031461.

[118] N. Saadallah, Hein Meling, and B. Daireaux. A simple machine in a complex environment: A petri net approach. In *Intelligent Engineering Systems (INES), 2011 15th IEEE International Conference on*, pages 387 –392, june 2011. doi: 10.1109/INES.2011.5954778.

[119] Nejm Saadallah and Benoit Daireaux. A goal based approach on top of Petri nets. In *International Workshop on Petri Nets and Software Engineering (PNSE2011)*.

[120] George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *STOC'77*, pages 61–76, 1977.

[121] Andrea Sackmann, Monika Heiner, and Ina Koch. Application of petri net based analysis techniques to signal transduction pathways. *BMC Bioinformatics*, 7:482, 2006.

[122] Andrea Sackmann, Dorota Formanowicz, Piotr Formanowicz, Ina Koch, and Jacek Blazewicz. An analysis of the petri net based model of the human body iron homeostasis process. *Computational Biology and Chemistry*, 31(1):1–10, 2007.

[123] J. C. Shepherdson and H. E. Sturgis. Computability of recursive functions. *J. ACM*, 10:217–255, April 1963. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321160.321170.

[124] J. Sifakis. Use of petri nets for performance evaluation. *Acta Cybernetica*, 4(2):185–202, 1979.

[125] Erlend skarsaune. Strike forces exploration rig closure.

[126] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[127] Jefferson W. Tester, Brian J. Anderson, Anthony S. Batchelor, David D. Blackwell, Ronald DiPippo, Elisabeth M. Drake, John Garnish, Bill Livesay, Michal C. Moore, Kenneth Nichols, Susan Petty, M. Nafi Toksoz, Ralph W. Veatch, Roy Baria, Chad Augustine, Enda Murphy, Petru Negraru, and Maria Richards. The future of geothermal energy; impact of enhanced geothermal systems egs on the united states in the 21st century. Technical Report INL/EXT-06-11746, Idaho National Laboratory, November 2006.

[128] Alexey Tovchigrechko. *Efficient symbolic analysis of bounded Petri nets using Interval Decision Diagrams*. PhD thesis, BTU Cottbus, Dep. of CS, October 2008.

[129] A. M. Turing. On Computable Numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42): 230–265, 1936.

[130] Antti Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65306-6. URL `http://dl.acm.org/citation.cfm?id=647444.727054`.

[131] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[132] Wil M. P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management*, pages 161–183, 2000.

[133] Wil M. P. van der Aalst. Making work flow: On the application of petri nets to business process management. In *ICATPN*, pages 1–22, 2002.

[134] W.M.P. van der Aalst. Putting high-level petri nets to work in industry. *Computers in Industry*, 25(1):45 – 54, 1994. ISSN 0166-3615. doi: 10.1016/0166-3615(94)90031-0.

[135] Ingolf Wassermann and Aravindh Kaniappan. How high-speed telemetry affects the drilling process. *JPT*, 61(6), 2009.

[136] John B. Watson. Psychology as the behaviorist views it. *Psychological Review*, 20:158–177, 1913. URL `http://psychclassics.yorku.ca/Watson/views.htm`.

[137] Wikipedia. Drilling rig. URL `http://en.wikipedia.org/wiki/Drilling_rig`.

[138] C. R. Zervos. *Coloured Petri Nets: their properties and applications*. PhD thesis, University of Michigan, 1977.

[139] M.C. Zhou and K. Venkatesh. *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Series in Intelligent Control and Intelligent Automation. World Scientific, 1999. ISBN 9789810230296.

[140] M.C. Zhou and K. Venkatesh. *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Series in Intelligent Control and Intelligent Automation. World Scientific, 1999. ISBN 9789810230296. URL `http://books.google.no/books?id=KQlXHGAQtCIC`.

[141] R. Zurawski and MengChu Zhou. Petri nets and industrial applications: A tutorial. *Industrial Electronics, IEEE Transactions on*, 41(6):567–583, December 1994.