The Symmetry Method for Coloured Petri Nets

- Theory, Tools and Practical Use

Louise Elgaard

PhD Dissertation

Department of Computer Science
University of Aarhus
Denmark

The Symmetry Method for Coloured Petri Nets - Theory, Tools and Practical Use

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Louise Elgaard
July 31, 2002

Preface

Short Summary in Danish

I dag indgår computer systemer i mange kritiske sammenhænge, f.eks. hospitalsudstyr, måleinstrumenter, fly og bilmotorer, hvor det er vigtigt at computer systemet virker som forventet. Indenfor datalogien er der i tidens løb udviklet flere metoder til at undersøge om computer systemer virker som de skal, men ikke alle metoder er lige anvendelige i praksis. Ph.d. afhandlingen "The Symmetry Method for Coloured Petri Nets – Theory, Tools, and Practical Use" beskæftiger sig med en sådan metode. Arbejdet tilstæber at udvikle teori og værktøjer, der gør at metoden bliver mere anvendelig i praksis. Ph.d. afhandlingen omfatter derfor to industrielle projekter hvor de udviklede teorier og værktøjer er anvendt og evalueret i industrielle sammenhænge.

Summary

A way to increase reliability of systems is to use *formal methods*, which are mathematically based methods for specifying and reasoning about systems. An example of a formal method for reasoning about systems is the state space method. The full *state space* of a system is a directed graph with a node for each reachable state of the system and an arc for each state change. From the full state space it is possible to verify whether the system satisfies a set of desired properties.

Several *reduction techniques* have been suggested for reducing the state space. With these reduction techniques only a subset of the full state space is represented or the full state space is represented in a compact form. An example of such a reduction technique is the *symmetry method*. The basic observation is that many distributed and concurrent systems posses a certain degree of symmetry, e.g., a system composed of identical components whose identities are interchangeable from a verification point of view. This kind of structural symmetry in the system is also reflected in the full state space of the system. The idea behind the symmetry method is to factor out this symmetry and obtain a *condensed state space* which is typically much smaller than the full state space, but from which the same kind of properties of the system can be derived without unfolding the condensed state space to the full state space.

State spaces and the symmetry method are not restricted to a specific modelling language. However, in the work presented in this thesis the symmetry

method is considered in the context of Coloured Petri Nets.

The thesis consists of two parts. Part I is the mandatory overview paper witch summarises the work which have been done. Part II is composed of four individual papers. Three of the papers have been published as conference papers in international conferences. One paper has been published as a workshop paper.

The overview paper introduces the symmetry method in the context of Coloured Petri Nets and summarises the four papers. An important part of the overview paper is a comparison of the work done with other research work within the field. This is done in form of a discussion of related work.

The first paper is based on an industrial cooperation project in which the symmetry method is put into practical use. The purpose of the project was to investigate the application of Coloured Petri Nets for validation of the communication protocols used in the Danish manufacturing company Danfoss. Analysis by means of state spaces successfully identified problems in the communication protocols and an alternative design was analysed using state spaces reduced by taking advantage of the inherent symmetries in the system. Exploiting the symmetries made it possible to analyse larger configurations of the system. The project also presents a first step towards improving the tool support for the symmetry method in form of a semi-automated consistency check, i.e., checking that the symmetries used for the reduction are symmetries that are actually present in the system.

A recurrent problem of the symmetry method is the high time complexity of the orbit problem, i.e., the problem of determining whether two states are symmetric. The second paper presents techniques to alleviate the negative impact of the orbit problem in state space generation with the symmetry method. The paper attacks the problem in two ways. Firstly, by presenting algorithms which exploit stabilizers of states, i.e., symmetries that map a state to itself, to potentially reduce the complexity of the orbit problem during state space generation. Secondly, by presenting a parallel version of the basic generation algorithm for state spaces reduced by means of symmetries. The techniques are implemented and evaluated on a number of practical experiments.

The third paper presents an important step towards making the symmetry method for Coloured Petri Nets applicable in practice. The paper presents the development of a tool which fully automates state space generation with the symmetry method. Prior tool support (the Design/CPN OE/OS Tool) required the user to implement two predicates determining whether two states/actions are symmetric or not. This requires both programming skills as well as a deep knowledge of the symmetry method. This is especially the case if the predicates are required to be efficient. When constructing condensed state spaces for CP-nets it can be observed that the predicates can be automatically deduced provided that the algebraic groups of permutations used for the reduction has been specified. The above observation motivated the construction of the tool which given an assignment of algebraic groups of permutations to the atomic colour sets of the CPN model automatically generates the predicates needed by the tool. During development of the tool different strategies for the predicates are investigated to see whether it is possible to develop general techniques which ensures an automatic but still efficient implementation of the predicates. The presented algorithms are implemented in the Design/CPN OPS tool and their applicability is evaluated based on practical experiments. The practical experiments show that the chosen strategies for the implementation of the predicates greatly influences whether the symmetry method is applicable in practice.

The fourth paper differ from the three first papers in the sense that it does not deal with the symmetry method. The paper is based on an industrial project. The paper presents results on the use of Coloured Petri Nets for the modelling and analysis of features and feature interactions in Nokia mobile phones. The paper presents how Coloured Petri Nets have been used to construct a model of parts of the software system in Nokia mobile phones. The paper is concerned with the interaction between features (the functionality of the mobile phone) implemented in individual modules. Feature interactions are investigated using simulations and state space analysis. The paper presents the Coloured Petri Nets model constructed in the project, describes how domain-specific graphics and Message Sequence Charts are used as an interface to simulations, and discusses how the project and in particular the construction of the model have influenced the development process of features in Nokia mobile phones.

Acknowledgements

During my PhD studies I have been associated to the Coloured Petri Nets Group (CPN Group) at the University of Aarhus. It has been a pleasant time which I have benefitted from both professionally and socially. For that I will express my thanks to both former and current members of the CPN group.

Especially, I would like to thank my adviser Søren Christensen for encouraging me to start my PhD studies and for guidance and support during the four years. Also thanks to Kurt Jensen for his involvement and guidance which I have highly appreciated.

Thanks to Lars M. Kristensen who besides co-authoring two of the papers taught me how to write research papers and from whom I have learned a lot. Thanks for support, help and encouragement during my PhD studies. Also thanks to the other three PhD students in the CPN group: Lisa Wells, Bo Lindstrøm and Thomas Mailund. We have often shared the same problems and concerns and I have appreciated the help and support from all of you.

During my PhD studies I have spent six months at Nokia Research Centre in Helsinki, Finland. Thanks to the Software Architecture Group headed by Juha Kuusela. The entire group has been very friendly and helpful and made my stay in Helsinki very pleasant. Especially thanks to Jianli Xu and Antti-Pekka Tuovinen who I worked closely together with and also co-authored one of the papers in this thesis. Also thanks to Francis Tam for introducing me to Finnish social life and letting me borrow his wonderful Finnish friends during my stay in Helsinki.

Finally, I would like to thank Kurt Jensen, Søren Christensen and Jacob Elgaard for reading and commenting on the overview paper in the thesis. The errors that may remain are entirely mine.

The work done for this thesis is supported by a grant from the Faculty of Science, University of Aarhus, Denmark.

Louise Elgaard,
Århus, Juli 31, 2002.

Contents

Preface **v**
Short Summary in Danish v
Summary v
Acknowledgements ix

I Overview **1**

1 Introduction **3**
1.1 General Introduction 3
1.2 Coloured Petri Nets and the Symmetry Method 4
1.2.1 Coloured Petri Nets for the Modelling of a Flowmeter System 5
1.2.2 Analysis by means of State Spaces with Symmetries . . . 7
1.3 Motivation and Aims of the Thesis 11
1.4 Outline and Structure of the Thesis 12
1.4.1 Reader's Guide 13

2 The Symmetry Method in Practice **15**
2.1 Background 15
2.2 Summary of Paper 16
2.3 Related Work 21

3 Exploting Stabilizers and Parallelism **25**
3.1 Background 25
3.2 Summary of Paper 26
3.3 Related Work 29

4 Algorithms and Tool Support for the Symmetry Method **33**
4.1 Background 33
4.2 Summary of Paper 34
4.3 Related Work 37

5 Modelling and Analysis of Feature Interactions **41**
5.1 Background 41
5.2 Summary of Paper 42
5.3 Related Work 48

6 Conclusions and Directions for Future Work 53
6.1 Summary of Contributions 53
6.2 Future Work 55

II Papers 59

7 Modelling and Analysis of a Danfoss Flowmeter System 61
7.1 Introduction 63
7.2 Overview of the Project 64
7.3 The Danfoss Flowmeter System 66
7.3.1 System Architecture and Communication Protocols . . . 66
7.3.2 CANAPP Design Patterns 68
7.4 CPN Model of the Flowmeter System 69
7.4.1 CPN Model Overview 70
7.4.2 Modelling of the CANAPPs 70
7.5 Analysis of two Initial Design Proposals 72
7.5.1 Analysis Goals 72
7.5.2 Analysis Results 73
7.6 Analysis of a third Design Proposal 77
7.6.1 Symmetry Specification 77
7.6.2 Consistency Check 79
7.6.3 Analysis Results 79
7.7 Conclusions 81

8 Exploiting Stabilizers and Parallelism 83
8.1 Introduction 85
8.2 Background 87
8.2.1 Coloured Petri Nets 87
8.2.2 The Symmetry Method 88
8.3 Canonical Representatives 90
8.4 Exploiting Stabilizers 91
8.4.1 Fewer Iterations 92
8.4.2 Fewer Markings Canonicalized 93
8.5 Exploiting Parallelism 94
8.6 Experimental Results 96
8.6.1 Stabilizer Algorithms 97
8.6.2 Parallel Algorithm 99
8.7 Conclusions and Related Work 100

9 State Space Generation with the Symmetry Method 103
9.1 Introduction 105
9.2 The Symmetry Method for CP-nets 107
9.2.1 Example: Distributed Database 107
9.2.2 Symmetry Specification 109
9.2.3 Restriction Sets 109
9.3 Condensed State Space Generation 110

9.4 Basic Algorithm for P_M . . . 112
9.4.1 Presentation of the Algorithm . . . 112
9.4.2 Experimental Results of the P_M^{Basic} Algorithm . . . 113
9.5 Approximation Techniques . . . 115
9.5.1 Presentation of the Algorithm . . . 115
9.5.2 Experimental Results of the P_M^{Approx} Algorithm . . . 118
9.6 Lazy Listing . . . 120
9.6.1 Presentation of the Algorithm . . . 120
9.6.2 Experimental Results of the $P_M^{Approx+Lazy}$ Algorithm . . . 123
9.7 Conclusions . . . 124

10 Modelling Features and Feature Interactions 127
10.1 Introduction . . . 129
10.2 The MAFIA Project . . . 130
10.3 The CPN Model . . . 132
10.3.1 Overview of the CPN model . . . 132
10.3.2 Modelling of the Features . . . 135
10.4 Simulations and Visualisation . . . 140
10.4.1 Animation of the display . . . 140
10.4.2 Controlling the simulations . . . 142
10.4.3 Message Sequence Charts . . . 142
10.5 Related and Future Work . . . 143
10.6 Conclusions . . . 145

Bibliography 149

Part I

Overview

Chapter 1

Introduction

This chapter gives an introduction to the symmetry method for Coloured Petri Nets and motivates the work presented in this PhD thesis within this research field. Section 1.1 gives a general introduction to and motivation for the symmetry method. Section 1.2 gives an informal introduction to Coloured Petri Nets and the symmetry method in the context of Coloured Petri Nets by means of an example. Section 1.3 presents motivations for and aims of the thesis. Finally, Sect. 1.4 gives an overview of the work included in the thesis and presents the structure of the rest of the thesis.

1.1 General Introduction

Today many computer systems are distributed and concurrent, ranging from small embedded systems in electronic equipment to large industrial production systems and geographically distributed systems. What is common for these systems is that the execution can proceed in many different ways depending on the individual components, their individual relative behaviour, and their interplay in the system. Therefore, it is extremely difficult to reason about the total behaviour of such systems and errors can go undetected for a long time. As we depend more and more on electronic systems in critical situations, e.g., in hospital equipment, traffic lights and car engines, the importance of being able to establish the correctness, or at least increase reliability of such systems is of great interest.

One way to increase reliability of systems is to use *formal methods*, which are mathematically based methods for specifying and reasoning about systems. An example of a formal method for reasoning about systems is the state space method. The full *state space* of a system is a directed graph with a node for each reachable state of the system and an arc for each state change. From the full state space it is possible to verify whether a system satisfies a set of desired properties, e.g., absence of deadlocks, the possibility to always reenter the system's initial state, etc. If a system does not posses the desired properties, then the full state space can be used to obtain counter examples, i.e., an execution of the system which leads to an undesired situation. This means that the state space can also be used to locate errors in the system. The main drawback in the practical use of the state space method is the *state explosion problem* [90]: the number of reachable states grows exponentially in the number of concurrent components, thus making it impossible to construct the full state space of the system.

Several *reduction techniques* have been suggested to alleviate the state explosion problem. With these reduction techniques only a subset of the full state space is represented or the full state space is represented in a compact form. An example of such a reduction technique is the *symmetry method* [22,23,47]. The basic observation is that many distributed and concurrent systems posses a certain degree of symmetry, e.g., a system composed of identical components whose identities are interchangeable from a verification point of view. This kind of structural symmetry in the system is also reflected in the full state space of the system. The idea behind the symmetry method is to factor out this symmetry and obtain a *condensed state space* which is typically much smaller than the full state space, but from which the same kind of properties of the system can be derived without unfolding the condensed state space to the full state space. The use of the symmetry method is highly dependent on the existence of supporting computer tools. Without suitable computer tools calculation, generation, and inspection of condensed state spaces is impossible for more than trivial systems.

State spaces and the symmetry method are not restricted to a specific modelling language [29]. However, in the work presented in this thesis the symmetry method is considered in the context of Coloured Petri Nets (CP-nets or CPNs) [46, 58]. CP-nets is a graphical modelling language based on Petri Nets [79] allowing modelling and formal analysis of distributed and concurrent systems. Design/CPN [19,50] is a computer tool supporting specification, simulation, and state space analysis of CP-nets. The tool is developed by the CPN Group at the University of Aarhus. In the area of CP-nets state spaces are also called occurrence graphs or reachability trees, state spaces with symmetries are also called occurrence graphs with symmetries and occurrence graphs with permutation symmetries. In this thesis the terms *state spaces* and *state spaces with symmetries* will be used.

1.2 Coloured Petri Nets and the Symmetry Method

In this section the concepts of CP-nets and the symmetry method for CP-nets [47,48] are informally introduced through a concrete example. The example considered is the Danfoss flowmeter system investigated in [65]. The work presented in [65] will be discussed in more detail in Chap. 2. The paper is included in its full version in Chap. 7. Section 1.2.1 contains an brief introduction to the Danfoss flowmeter system, followed by an informal introduction to CP-nets using a CPN model of the flowmeter system. Section 1.2.2 presents the ideas of state spaces and state spaces with symmetries.

1.2.1 Coloured Petri Nets for the Modelling of a Flowmeter System

CP-nets have proven powerful for modelling of concurrent systems [49]. An example of a concurrent system is the Danfoss flowmeter system which will briefly be introduced in the following.

Flowmeters are primarily used to make measurements on the flow of liquid through pipes. The flowmeter system studied in [65] consisted of several processes each conducting measurements on the flow of liquid. Examples are processes measuring the amount of water flowing through a pipe, processes measuring the temperature of the water, and processes doing calculations based on measurements obtained by other processes. Figure 1.1 illustrates the overall architecture of a flowmeter system. A flowmeter system consists of one or more *modules* connected via a *Controller Area Network* (CAN) [62]. Each module consists of a number of processes called *CAN Applications* (CANAPPs) and a *driver* that interfaces the module to the CAN. Figure 1.1 shows an example of a flowmeter system consisting of three modules containing two, three, and four CANAPPs, respectively. Each CANAPP in the system has a local memory which holds a number of *attributes*. The communication in the system consists of asynchronous message passing between the CANAPPs. This message passing allows each CANAPP to read and write the attributes of the other CANAPPs.

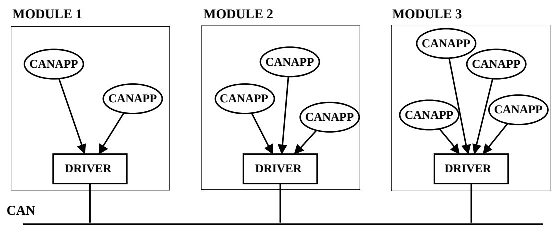


Figure 1.1: Overall architecture of a flowmeter system.

In the following an informal introduction to CP-nets is given using the CPN model of the flowmeter system created in the project reported on in [65] as an example. This section will informally introduce CP-nets as they are formally defined in [46] as well as the style in which CP-nets appear in the rest of this thesis. A more detailed introduction to CP-nets can be found in [58]. The CPN model is created in the Design/CPN tool [26] and the introduction given here reflects the terminology used in the Design/CPN tool. A CP-net is hierarchically structured into modules (subnets), also referred to as *pages* in Design/CPN terminology. Figure 1.2 gives an overview of the CPN model of the flowmeter system by showing how it has been hierarchically structured into 12 pages. Each node in Fig. 1.2 represents a page of the CPN model. An arc between two nodes indicates that the source node contains a so-called *substitution transition* whose behaviour is described on the page represented by the destination node.

A CP-net is created as a graphical drawing with textual inscriptions. In

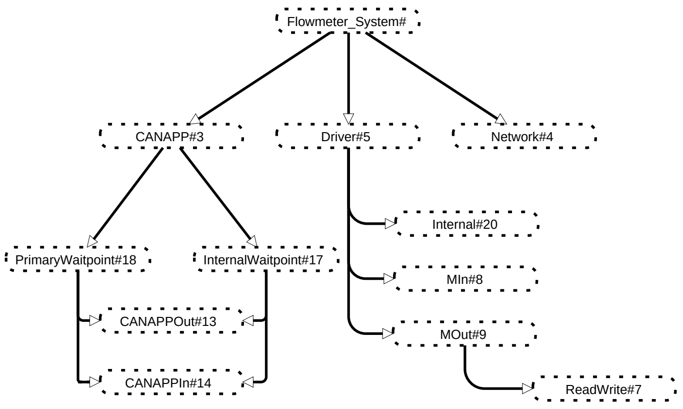


Figure 1.2: The hierarchy page.

the following it is shown how CANAPPs are modelled as a CP-net. Figure 1.3 depicts the page CANAPPOut, which models the part of a CANAPP generating requests to the other CANAPPs and awaiting responses.

In contrast to many other modelling languages CP-nets are both state and action oriented. A state of a CP-net is represented by means of *places* which are drawn as ellipses with a name positioned inside. The places contain *tokens*, which carry data values. The data values are in CPN terminology referred to as *colours*. Each place has a type, in CPN terminology referred to as *colour set* that determines the kind of tokens that can reside on the place. The colour set of a place is written next to the place. A *state* of a CP-net is a distribution of tokens on the individual places. A state is in CPN terminology also referred to as a *marking*. A CP-net has a distinguished marking called the *initial marking*, corresponding to the initial state of the system.

In Fig. 1.3 the two places RequestOut and ResponseIn both have the colour set

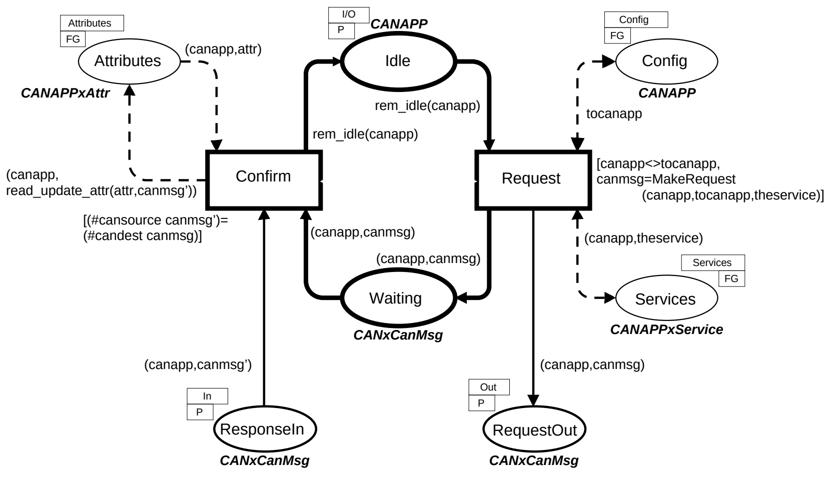


Figure 1.3: The page CANAPPOut.

CANxCanMsg, which denotes the cartesian product of CANAPPs and messages. These two places model the buffers used to temporarily store outgoing requests to other CANAPPs in the system and incoming responses to previously sent requests. The places Idle and Waiting model the control flow in the part of a CANAPP generating requests to the other CANAPPs and awaiting responses. The place Attributes is used to model the attributes of the CANAPP. The places Config and Services are used to model configuration information (information about other CANAPPs in the flowmeter system) which can be accessed by the CANAPP.

Actions of a CP-net are represented as *transitions* which are drawn as rectangles with a name positioned inside. The transitions and places of a CP-net are connected by *arcs*. Transitions remove tokens from the places connected to incoming arcs and add tokens to the places connected to outgoing arcs. The tokens removed and added are determined by *arc expressions* which are inscriptions positioned next to the arcs. In the Design/CPN tool these inscriptions are written in the Standard ML programming language [70]. In Fig. 1.3 the sending of a request is modelled by the transition Request, which causes the CANAPP to change its state from being Idle to Waiting and passes the message to the driver by putting it into the buffer modelled by the place RequestOut. The token put on place RequestOut is specified by the expression (canapp,canmsg) which is the cartesian product of a CANAPP and a message.

A transition can remove and add tokens when two conditions are fulfilled. Firstly, sufficient tokens must be present on the places connected to incoming arcs, i.e., it must be possible to assign data values to the *free variables* of the transition in such a way that each arc expression *evaluate* to a multi-set of tokens which is a subset of the tokens present on the corresponding input place. Secondly, a boolean expression assigned to the transition called a *guard* must evaluate to true. When these two conditions are fulfilled the transition is said to be *enabled* and it may *occur*, thereby removing tokens from the input places and adding tokens on the output places of the transition. A pair consisting of a transition and an assignment of data values to the free variables of the transition is called a *binding element*.

When a message is put in the buffer modelled by the place RequestOut, the driver in the module will remove the message from the place RequestOut and deliver it to the destination. When the response returns, the corresponding message is put in the buffer modelled by the place ResponseIn. The actual reception of a response is modelled by the transition Confirm. An occurrence of this transition removes the response from the place ResponseIn, updates the attributes of the CANAPP (modelled by place Attributes) and causes the CANAPP to change its state from Waiting to Idle.

1.2.2 Analysis by means of State Spaces with Symmetries

As seen above CP-nets have a graphical representation. Furthermore, CP-nets have a formally defined semantics allowing formal analysis [46]. The CPN model introduced above can therefore be used to analyse whether the CPN model of the flowmeter system possesses a number of desired properties.

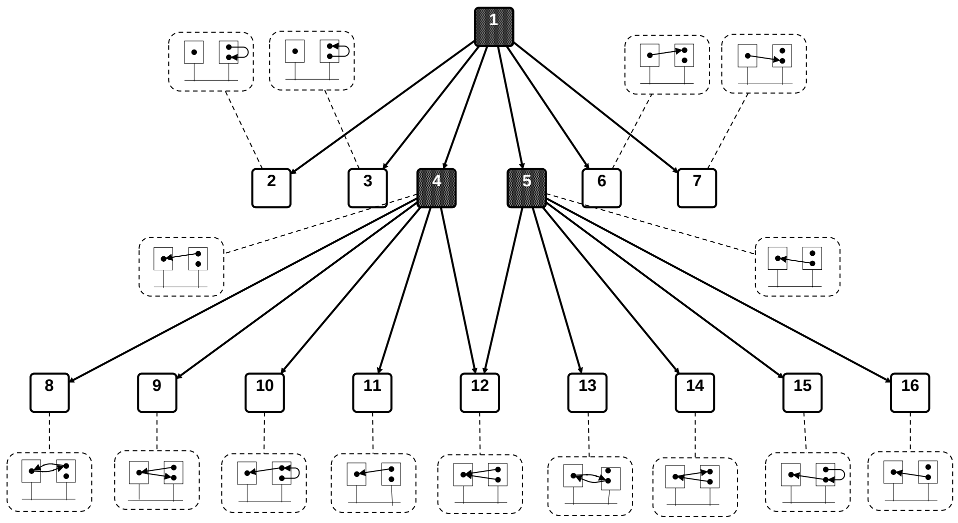
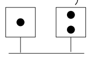
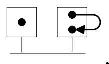


Figure 1.4: Initial fragment of the full state space.

A *full state space* [46] of a CP-net is a directed graph with a node for each reachable state of the CP-net and an arc for each state change of the CP-net. Figure 1.4 shows the initial fragment of the full state space for a flowmeter system consisting of two modules containing one and two CANAPPs, respectively. This system configuration is graphically represented as . The nodes in the state space coloured black are the nodes which are fully explored, i.e., the nodes for which all successors have been calculated and included in the state space. Modules are numbered from left to right, and the CANAPPs in a module from top to bottom. The notation CANAPP$_{(i,j)}$ is used to denote CANAPP j in module i. Similarly, a communication in the system will be graphically represented. A situation where CANAPP$_{(2,1)}$ has initiated request towards CANAPP$_{(2,2)}$ will be represented as .

Node 1 in Fig. 1.4 corresponds to the initial marking/state and has six immediate successor nodes corresponding to the possible requests which can be initiated in the system (each CANAPP can initiate a request to the two other CANAPPs in the flowmeter system). Only the successor nodes of node 4 and 5 are shown in the layer of the state space corresponding to two steps from the initial state. For each of the nodes it is indicated in the associated dashed box what communication have been initiated, e.g., node 4 corresponds to a state of the system in which CANAPP$_{(2,1)}$ has initiated a request towards CANAPP$_{(1,1)}$.

Even for small CP-nets such a full state space may become very large, thus encountering the *state explosion problem* [90], i.e., the state space starts to grow rapidly when analysing systems of increasing size, e.g., flowmeter systems with an increasing number of CANAPPs. In the flowmeter system CANAPPs located in the same module can be considered interchangeable/symmetric, i.e.,

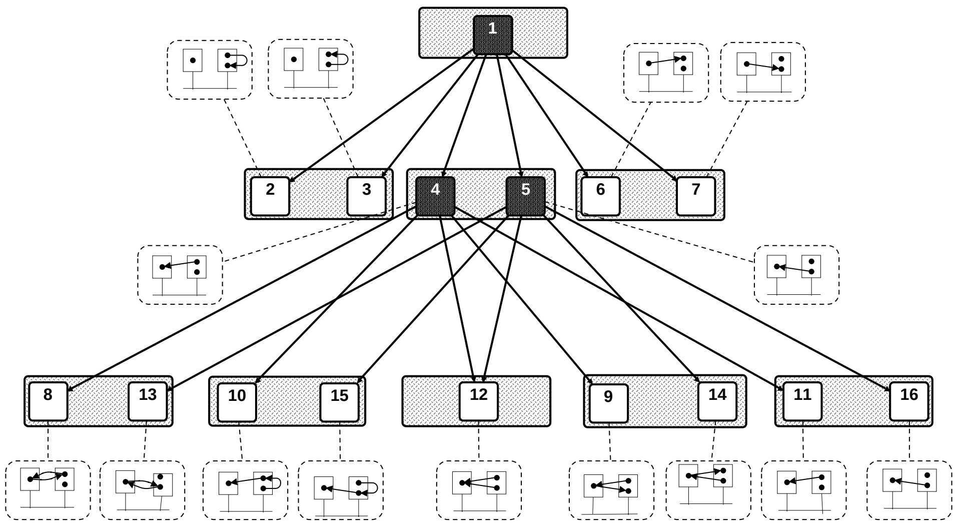


Figure 1.5: Reorganised initial fragment of the full state space. The grey boxes indicate symmetric markings.

CANAPP$_{(2,1)}$ and CANAPP$_{(2,2)}$ are symmetric seen from a verification point of view. Also modules containing the same number of CANAPPs can be considered symmetric. There is, however, no such modules in the configuration used as example. When considering the state space of the flowmeter system it can be seen that this kind of structural symmetry also is reflected in the state space of the system. Node 4 in Fig. 1.4 corresponds to a marking where CANAPP$_{(2,1)}$ has initiated a request to CANAPP$_{(1,1)}$, while node 5 corresponds to a marking where CANAPP$_{(2,2)}$ has initiated a request to CANAPP$_{(1,1)}$. For node n the corresponding marking is denoted M_n. M_4 and M_5 can be obtained from each other by a permutation which swaps the identity of CANAPP$_{(2,1)}$ and CANAPP$_{(2,2)}$, hence M_4 and M_5 can be considered symmetric.

Figure 1.5 shows the same fragment of the full state space as Fig. 1.4, but in Fig. 1.5 the states are reorganised such that symmetric markings are positioned next to each other in grey boxes. The grey boxes indicate equivalence classes of symmetric markings. The first box represents the equivalence class containing the initial marking only, the second box represents the equivalence class containing the markings M_2 and M_3, the third box represents the equivalence class containing the markings M_4 and M_5, and the fourth box represents the equivalence class containing M_6 and M_7. The successors of M_4 and M_5 are grouped into equivalence classes in a similar way. From Fig. 1.5 it can be seen that two symmetric markings, such as M_4 and M_5, have symmetric successor markings. Hence, for a set of symmetric markings it is sufficient to explore the possible behaviours of the system for only one of these markings.

The basic idea behind the symmetry method is to lump together symmetric sets of markings/binding elements into equivalence classes as shown in Fig. 1.5. It is sufficient to store a single representative from each equivalence class, e.g., the marking M_4 can be chosen as a representative for the equivalence class

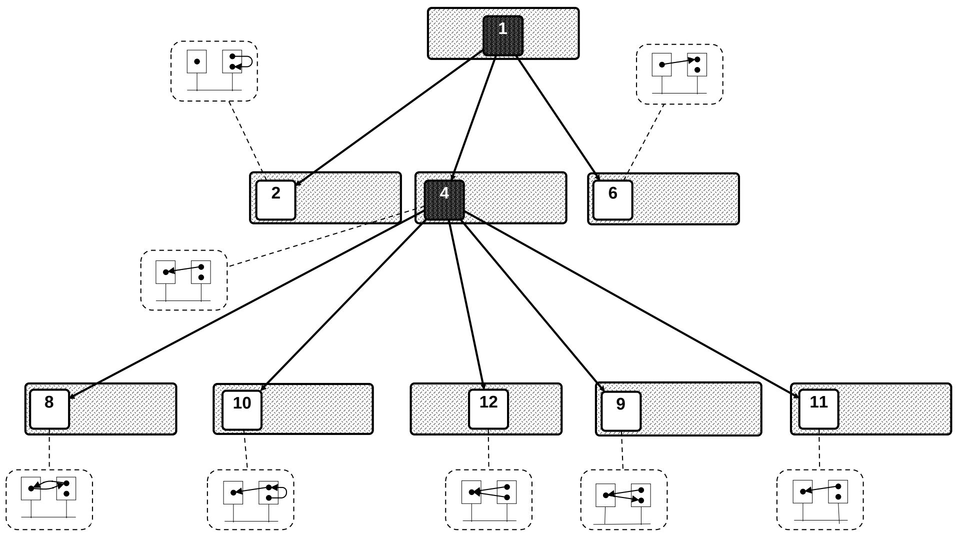
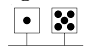


Figure 1.6: Initial fragment of the condensed state space containing one representative from each equivalence class.

containing the markings M_4 and M_5. In this way a *condensed state space* is obtained which is typically much smaller than the full state space. The initial fragment of the condensed state space for the flowmeter system is shown in Fig. 1.6. From Fig. 1.6 it can be seen that only one marking from each equivalence class is included in the condensed state space. The total number of markings reachable within two steps from the initial marking is 23. The total number of equivalence classes reachable within two steps from the initial marking is 13. Hence it is possible to represent the 23 markings which can be reached in two steps from the initial marking using only 13 nodes in the condensed state space. These number may not at first sight seem impressive. However, in general the highest possible reduction for a configuration is $\Pi_{n \in \mathbb{N}}(n! \cdot \text{the number of modules containing } n \text{ CANAPPs})$. Hence, the highest possible reduction for the configuration  is 120. Practical experiments show that the reduction obtained is very close to the highest theoretically possible.

The symmetries used for the reduction are obtained from permutations of the atomic colours in the CP-nets. Hence, in the rest of the thesis such symmetries are denoted *permutation symmetries* and a condensed state space is also called a *state space with permutation symmetries* (SSPS). The permutation symmetries used for the reduction must be chosen in such a way that they capture the symmetries actually present in the system. When this is the case the choice of symmetries are said to be *consistent* [47]. Consistency of the choice of symmetries ensures that the full state space can be constructed from the SSPS. When the choice of symmetries used for the reduction is consistent the SSPS has an arc from one equivalence class to another if and only if there is an arc from a node in the first equivalence class leading to a node in the second equivalence class. The fact that symmetric markings have symmetric sets of successor markings and binding elements ensures that either all or none of the markings in the first equivalence class have an enabled binding element leading to a marking in the second equivalence class.

1.3 Motivation and Aims of the Thesis

The symmetry method is a general state space reduction technique and is as such not restricted to a certain modelling language. Several variants have been suggested for reasoning about different classes of properties, e.g., for safety properties [44], for temporal logics formulae specified in CTL* [23, 24, 28] and LTL [30, 38], for properties of P/T nets [85], and in the context of CP-nets a variant for reasoning about standard dynamic properties of CP-nets [47, 48].

One of the common problems for all the variants of the symmetry method is how to determine whether two states are symmetric. A selection of papers, e.g., [55, 80, 81] presents results and techniques for this problem for low level Petri Nets.

In the context of CP-nets the theory of the symmetry method is well developed [47, 48]. However, no papers document the use of the symmetry method in practice in the context of Coloured Petri Nets. A first step toward making the symmetry method for Coloured Petri Nets applicable in practice is made in 1996 where the Design/CPN tool was extended with support for generation of SSPSs. The tool is called the Design/CPN OE/OS tool [52] and is together with a case study documented in [53]. However, the tool requires the user to implement two predicates expressing whether two states/actions are symmetric. Hence, the tool does not address the problem of determining whether two states are symmetric, i.e., the orbit problem. [4, 32, 51] all presents ideas for efficient solutions to the orbit problem for CP-nets but the ideas are not integrated into state space generation and in [4, 32] the ideas are not evaluated in practice.

The above motivated the research work done as a part of this thesis. Early experimental results [65] show that in practice an efficient implementation of the orbit problem is crucial for whether the symmetry method is applicable in practice for larger models, i.e., models with a potentially large number of symmetries. Furthermore, a basic observation in the symmetry method for CP-nets is that when the symmetries of the system are known, the problem of determining whether two states are symmetric can be performed fully automatic. This motivates development for general algorithms and techniques for the orbit problem as well as tool support such that the techniques and ideas can be integrated into state space generation without manual implementation or other (model specific) user invention.

One of the key issues motivating the work done in this thesis is to develop techniques and tools which make the symmetry method for CP-nets applicable in practice. Throughout my PhD studies the work has included three aspects of the symmetry method of Coloured Petri Nets: *Theory*, *tools*, and *practical use.* I find that all three aspects and the interplay between them are important aspects towards making the symmetry method for CP-nets applicable in practice. New ideas and techniques are developed during research work, these are applied and explored in small examples and later in industrial settings. This may again motivate further development of the techniques. The applicability of the developed techniques is closely related to the existence of suitable computer tools. Without proper computer support it is impossible to calculate and use SSPSs for more than small examples. This should imply that neither of the three aspects: theory, tools and practical use should be treated in isolation.

Theory. The research work concentrates on the development of general techniques and algorithms for the orbit problem of CP-nets. The algorithms are integrated into state space generation for CP-nets.

Tools. The developed algorithms are implemented in a tool which automates the symmetry method for CP-nets thus making it possible to automatically calculate the SSPSs of a CP-net with when the user has specified the permutation symmetries used for the reduction.

Practical Use. The developed algorithms and tools are evaluated in a number of practical experiments.

1.4 Outline and Structure of the Thesis

The thesis is divided into two parts. Part I is the mandatory overview paper which summarises the work which have been done. Part II is composed of four individual papers. Three of the papers [65–67] have been published as conference papers in international conferences. One paper [63] has been published as a workshop paper. Three of the papers [63, 65, 66] document work with the symmetry method for CP-nets. The fourth paper [67] document work with application of CP-nets in an industrial setting.

Chapter 2 summarises the paper *Modelling and Analysis of a Danfoss Flowmeter System using Coloured Petri Nets* [65]. The paper presents joint work with Lars M. Kristensen and has been published in M. Nielsen and D. Simpson, editors, *Proceedings of the 21th International Conference on Application and Theory of Petri Nets*, volume 1825 of Lecture Notes in Computer Science, pages 346–366, Springer-Verlag, 2000. The paper is contained in full in Chapter 7.

Chapter 3 summarises the paper *Exploiting Stabilizers and Parallelism in State Space Generation with the Symmetry Method* [66]. The paper presents joint work with Lars M. Kristensen and has been published in *Proceedings of the Second International Conference on Application of Concurrency to System Design*, IEEE, 2001. The paper is contained in full in Chapter 8.

Chapter 4 summarises the paper *Coloured Petri Nets and State Space Generation with the Symmetry Method* [63]. The paper has been published in K. Jensen, editor, *Proceedings of the Fourth Workshop on Applications of Coloured Petri Nets and CPN/Tools*, Department of Computer Science, University of Aarhus, Denmark, 2002. The paper is contained in full in Chapter 9.

Chapter 5 summarises the paper *Modelling of Features and Feature Interactions in Nokia Mobile Phones using Coloured Petri Nets* [67]. The paper presents joint work with Antti-Pekka Tuovinen and Jianli Xu and has been published in J. Esparza and C. Lakos, editors, *Proceedings of the 23rd International Conference on Application and Theory of Petri Nets*, volume 2360 of Lecture Notes in Computer Science, pages 294–313, Springer-Verlag, 2002. The paper is contained in full in Chapter 10.

Chapter 2–5 are all divided into three sections; **Background** gives a brief introduction to and some background for the topic of the paper, **Summary of Paper** gives a brief summary of the main conclusions presented in the paper, and **Related Work** discusses and compares the paper to related work. Chapter 6 concludes the work presented in this thesis and discusses some directions and ideas for future work.

1.4.1 Reader's Guide

The reader of this thesis is assumed to be familiar with the concepts of Petri Nets. CP-nets are informally introduced in Chapter 1 and further knowledge of CP-nets is not a prerequisite. However, more knowledge may be an advantage for the reader who wants to study the symmetry method for Coloured Petri Nets in more detail. For an introduction to CP-nets [58] is recommended as a starting point, a formal definition of CP-nets can be found in [46], and a formal definition of the symmetry method for CP-nets can be found in [47, 48].

The order in which the four papers [63, 65–67] are included in this thesis is influenced by part I of the thesis. The presentation of the papers is structured in the order which feels most natural for the thesis as a whole. For the reader who is only interested in parts of the thesis the following guidelines are given: Chapter 1 contains an introduction to the research field and can be skipped by readers who are already familiar with CP-nets and the symmetry method for CP-nets. Chapter 2-4 presents work on the symmetry method. The reader is recommended to read Chapter 3 before Chapter 4. Chapter 5 presents work on the application of Coloured Petri Nets to feature interactions in mobile phone software and can be read independently of the other chapters.

Chapter 2

The Symmetry Method in Practice: Analysis of a Flowmeter System

This chapter discusses the paper *Modelling and Analysis of a Danfoss Flowmeter System using Coloured Petri Nets* [65]. Section 2.1 describes the background of the paper and contains an introduction to the results presented in the paper. Section 2.2 gives a summary of the paper and discusses the main results of the paper. Finally, Sect. 2.3 contains a discussion of related work.

2.1 Background

The paper *Modelling and Analysis of a Danfoss Flowmeter System using Coloured Petri Nets* [65] presents an industrial cooperation project where CP-nets and state space methods are applied in practice for the modelling and analysis of a real product: a flowmeter system from the Danish manufacturing company Danfoss. The project was carried out as a joint project between Danfoss Instrumentation (a subgroup of the Danfoss company) and the CPN group at the University of Aarhus. In the rest of the chapter the project is referred to as the *Danfoss project*.

The aim of the project was to investigate the application of CP-nets for validation of the communication protocol used in the flowmeter system; a topic of increasing interest within Danfoss due to realised problems, e.g., deadlocks found via practical tests. Before the project started two different communication strategies (in [65] denoted design alternatives) were suggested for the communication protocol used in the flowmeter system. The project aims at investigating both communication strategies using CP-nets and its supporting Design/CPN tool. The CPN model created captures both communication strategies. The strategy used as well as the configuration of the flowmeter system analysed is determined by the initial marking.

The validation in the project is done using state spaces as a formal analysis method. A state space of the CPN model created in the project contains a node for each of the reachable states of the CPN model of the flowmeter system and an arc for each state change. From the state spaces constructed it can be verified whether the CPN model (with the chosen communication strategy and configuration) possesses a number of desired properties. If the properties are violated the state space can be used to obtain counter examples, i.e., a path of states/actions leading to a state where a property is violated.

The project extends the analysis results using state spaces with permutation symmetries (SSPS) to be able to analyse larger configurations of the flowmeter system. In the construction of SSPSs CANAPPs located in the same module are considered symmetric as well as modules containing the same number of CANAPPs are considered symmetric. By exploiting this symmetry in the construction of the state spaces it is possible to obtain a significant reduction in the number of states and actions included in the state spaces. Hence, making it possible to potentially analyse larger configurations of the CPN model of the flowmeter system.

The project group consisted of both members from Danfoss and from the CPN group, i.e., both members with expertise in the domain (the flowmeter system) and members with expertise in the methods and tools to be applied. The members from Danfoss had beside a basic familiarity of the concepts of low level Petri Nets no prior knowledge of Coloured Petri Nets, modelling or formal analysis. Hence, it was decided to visualise the communication between CANAPPs in the CPN model of the flowmeter system using Message Sequence Charts (MSCs) [45].

2.2 Summary of Paper

The project was structured into three phases: modelling, state space analysis, and analysis by means of state spaces with permutation symmetries. The main contribution of the paper is the application of state spaces and state spaces with permutation symmetries to a large CPN model developed in an industrial setting. Hence, the discussions in this chapter focus on the two last phases constituting the formal analysis of the CPN model of the flowmeter system.

Modelling of the Flowmeter System. The CPN model of the communication protocol in the flowmeter system created in the project is a hierarchical CPN model developed in the Design/CPN tool [19, 26].

The flowmeter system and parts of the CPN model created in the project are presented in Chapter 1 where it was used to introduce CP-nets and the symmetry method for CP-nets. The construction of the CPN model or the CPN model itself will therefore not be discussed in more detail. More details can be found in [65] which is contained in full in Chapter 7.

The CPN model is validated using simulations; both interactive (step-by-step) simulations and later (when reliability of the model is increased) more automatic simulations. For that purpose the CPN model is extended to automatically produce MSCs from simulations which allow the communication between the CANAPPs in the flowmeter system to be observed at a more high level than observing the token game in the CP-net. MSCs are graphical drawings showing vertical lines corresponding to processes in the system (CANAPPs in the flowmeter system). Arrows between the vertical lines correspond to messages sent in the system. Examples of MSCs are given in [65]. Since the MSCs are constructed automatically from simulations of the CPN model, the simulations can be observed without inspecting the underlying CP-nets. Furthermore, the MSCs are used to visualise executions of the CPN model leading to error states found via state space analysis.

State Space Analysis. After the modelling phase of the project the desired properties of a flowmeter system were formulated to serve as a basis for the formal analysis. Both of the communication strategies included in the CPN model of the flowmeter system were analysed to see whether they fulfil the properties. The analysis was performed by means of state spaces using the Design/CPN OG tool [17]. State spaces were chosen as the analysis method for two reasons. Firstly, the tool support for state space generation and analysis is well developed in the Design/CPN tool. Secondly, one of the aims of the project was to demonstrate the use of state spaces and the Design/CPN OG tool in industrial settings.

To make it possible to analyse whether the model satisfies the properties, the properties are translated into dynamic properties of the CPN model. This makes it possible to formulate the requirements as queries in the Design/CPN OG tool. Below an informal description of each of the properties are given followed by a translation of the properties into properties of the CPN model of the flowmeter system.

Absence of Deadlocks.

This property expresses that it should not be possible to bring the flowmeter system into a deadlock situation, i.e., a state where all the processes (CANAPPs) are blocked.

At the level of the CPN model this property can be expressed as absence of dead markings, i.e., markings without enabled binding elements. This is a standard dynamic property of a CPN model and the OG tool has a build in function `listdeadmarkings()`, which find the dead markings (if any) in a state space.

Absence of Attribute Corruption.

This property expresses that when a CANAPP has initiated a request its attributes must not be changed before the request has been completed.

At the level of the CPN model the property can be formulated using temporal logics [22]. The Design/CPN OG tool library ASK-CTL [20] makes it possible to make queries expressed in a state and action oriented variant of CTL [12] which exploits both the state (markings) and action (binding elements) oriented nature of CP-nets. The absence of attribute corruption for a given CANAPP$_{(i,j)}$ can be expressed as the following action-based CTL formula. The formula is explained in detail below.

$$\begin{aligned}&\mathbf{AG}((\mathsf{Request}, \langle \mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)}\rangle) \Rightarrow \\ &\quad \mathbf{A}((\neg(\mathsf{Indication}, \langle \mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)}\rangle))\ \mathbf{U}\ (\mathsf{Confirm}, \langle \mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)}\rangle)))\end{aligned}$$

The formula states that whenever (denoted **AG**) the transition Request occurs in a binding corresponding to CANAPP$_{(i,j)}$, then in all futures (denoted **A**) the transition Indication cannot occur in a binding corresponding to CANAPP$_{(i,j)}$ until (denoted **U**) the transition Confirm has occurred in a binding corresponding to CANAPP$_{(i,j)}$. An occurrence of the transition Request in a binding corresponding to CANAPP$_{(i,j)}$ has been written as (Request,⟨canapp=CANAPP$_{(i,j)}$⟩). The binding of Indication and Confirm is written in a similar way. An occurrence of the transition Request (see Fig. 1.3) models the start of a request, an occurrence of transition Indication (not shown in a figure) models the start of handling an incoming request, and an occurrence of transition Confirm (see Fig. 1.3) models the reception of a response.

Topology Independence.

The last property states that the two first properties must be valid independent of the configuration of the flowmeter system.

At the level of the CPN model this property can be investigated by analysing different configurations of the flowmeter system. Investigating different configurations can be done by simply changing the initial marking of the CPN model. Using state spaces topology independence cannot be completely verified but only verified for the configurations for which the state space has been constructed. Hence, investigating different configurations of the CPN model of the flowmeter system will not result in a complete verification of this property but solely increase the reliability of the CPN model. This issue will be touch in in the discussion of related work.

Using the Design/CPN OG tool state spaces were calculated and analysed for a number of configurations of the flowmeter system using both communication strategies. The concrete list of the configurations analysed as well as the generation statistics can be found in [65] which is contained in full in Chapter 7. In this phase of the project the *state explosion problem* [90], i.e., the size of the state spaces start to grow rapidly when system parameters increase, was encountered. However, even in the small configurations analysed errors were found, i.e., the desired properties of a flowmeter system were not fulfilled. In all configurations analysed one of the communication strategies failed to fulfil the *absence of deadlocks*, the other communication strategy failed to fulfil the *absence of attribute corruption*.

Examples of paths[1] leading to states where the properties were not fulfilled were inspected and discussed within the project group in order to identify and understand the shortcomings of the communication strategies. For presentation purposes and to improve the readability of the execution paths leading to error states found via state space analysis the paths were visualised using the MSCs like the MSCs used to visualise simulations of the CPN model.

Analysis by means of State Spaces with Permutation Symmetries. On the basis of the analysis results obtained in the second phase of the project

[1]The Design/CPN tool has a build-in function which returns one of the shortest paths leading from a marking to another.

it was concluded that another communication strategy than the two communication strategies included in the CPN model and analysed in the second phase of the project was needed. Based on the analysis results from the second phase of the project a new communication strategy was designed. The CPN model was revised to capture the new communication strategy and for small configurations it was verified that the revised CPN model fulfilled the two properties: *absence of deadlocks* and *absence of attribute corruption*. Again the state explosion problem prevented analysis of larger configurations of the flowmeter system. Since the flowmeter system is composed of a number of identical components, i.e., CANAPPs, whose behaviour are identical it was decided to use the symmetry method in the analysis of the CPN model of the flowmeter system. However, for the symmetry method to apply it need to be ensured that the symmetry method preserves the properties that have to be verified.

Absence of Deadlocks.

A state in a SSPS corresponds to an equivalence class of states in the state space. The symmetry specification used for the reduction is required to be consistent, i.e., fulfil the three requirements in Def. 3.16 in [47]. Consistency of the symmetry specification ensures that symmetric states all have symmetric enabled binding elements as well as symmetric successor states. Hence, a dead state in the SSPS corresponds to an equivalence class of symmetric dead states (markings) in the state space and vice versa. Hence, absence of deadlocks is preserved by the symmetry method.

Absence of Attribute Corruption.

To reflect the new communication strategy analysed the CPN model was changed. Hence, the action-based CTL formula used for the analysis in the previous phase of the project need to be changed accordingly. The query is changed to take into account that in the revised communication strategy the transition Indication can occur in two modes (accept or reject) depending on whether the request is accepted or rejected by the receiving CANAPP. The new formula can be expressed as the following action based CTL formula.

$$\begin{aligned}&\mathbf{AG}((\mathsf{Request},\langle\mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)}\rangle) \Rightarrow\\ &\qquad\mathbf{A}((\neg(\mathsf{Indication},\langle\mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)},\mathsf{mode}{=}\mathsf{accept}\rangle))\ \mathbf{U}\\ &\qquad\ (\mathsf{Confirm},\langle\mathsf{canapp}{=}\mathrm{CANAPP}_{(i,j)}\rangle)))\end{aligned}$$

The action-based CTL formula above expresses absence of attribute corruption for a specific $\mathrm{CANAPP}_{(i,j)}$, i.e., the formula refers to an occurrence sequence related to a specific $\mathrm{CANAPP}_{(i,j)}$. Since all permutations of CANAPPs are allowed this property is not a priory preserved by the symmetry method. However, in [23, 28] it is shown that the symmetry method preserves the truth value of a CTL formula if the truth value of its atomic state propositions are invariant under the permutation symmetries. Hence, the property can only be verified for CANAPPs which are not allowed to be permuted by the symmetry specification. Such CANAPPs are present in configurations of the flowmeter system containing a module with a single CANAPP. Another possibility is to strengthen the symmetry specification such that the CANAPP in question cannot be permuted. The experimental results presented in [65] is obtained in configurations containing a module with a single CANAPP. The reason for this choice was that strengthening the symmetry specification required a re-implementation of the predicates expressing whether two markings/binding elements are symmetric; a quite cumbersome task when implementing the predicates by hand. In Chapter 4 the construction of a tool which automates the implementation of the predicates is discussed. When using this tool experiments with different symmetry specifications, e.g., different kind of strengthened symmetry specifications, can easily be performed without a manual re-implementation of the predicates.

Topology Independence.
This property is also in this case analysed by investigating different configurations of the flowmeter system. The configuration is determined by the initial marking and in each case it must be ensured that the initial marking, M_0, fulfils the requirement for consistency, i.e., $\forall \phi \in \Phi : \phi(M_0) = M_0$.

The analysis was done using the Design/CPN OE/OS tool [52] which supports generation and analysis of SSPSs. However, the calculation of SSPSs are not fully automatic; instead of assigning symmetry groups of permutations to the atomic colour sets of the CP-net the user of the tool is required to implement two predicates expressing when two markings/binding elements are symmetric. Furthermore, since the predicates are user supplied, it is also the responsibility of the user to ensure that the predicates implements a consistent symmetry specification, i.e., that the symmetries used for the reduction is actually symmetries present in the CPN model.

As a part of the project an extension to the OE/OS tool was developed which supports a semi automatic check for consistency. Consistency of the symmetry specification ensures that symmetric markings have symmetric sets of enabled binding elements and symmetric sets of successor markings. Def. 3.16 in [47] formally defines three requirements that must be fulfilled for a symmetry specification to be consistent. Informally the definition states that: 1) each symmetry given by the symmetry specification must map the initial marking to itself, 2) all transitions have symmetric guards, i.e., treat symmetry colours the same way, and 3) all arc expressions and symmetries given by the symmetry specification commutes. It is worth noticing that the three requirements are structural and therefore can be checked without calculating the occurrence sequences of the CP-net. Hence, the check for consistency is invoked independently of the state space generation. Since consistency ensures that the dynamic properties of the CP-net can be derived from the SSPSs the condensed state space is only calculated and analysed if the symmetry specification is found to be consistent.

The tool extension checks that the three requirements are fulfilled. 1) can be checked by an iteration of all symmetries given by the symmetry specification. In practice this approach may be very time-consuming, or even impossible, when the number of symmetries allowed by the symmetry specification is large. Using the techniques presented in Chapter 3 the check can be implemented more efficiently. 2) and 3) are checked by considering each guard and arc expression of the CP-net in turn. The check is based on a combination of semantic and syntactical checks. Some guards/arc expressions have syntactic restrictions that ensure that the consistency requirement is fulfilled, e.g., simple patterns like variable names. Experimental results show that it is only the case for very few guards/arc expressions that consistency cannot be determined using the structural check. Only in that case a semantic check is performed. The semantic check is based on evaluating the guard/arc expression in all possible bindings. As a consequence the semantic check is very time consuming but together with the syntactic check it is possible to make a fully automatic check for consistency for the CPN model of the flowmeter system.

2.3 Related Work

The main contribution of the paper is the application of state spaces and SSPSs and the supporting tools in an industrial setting. In the following the results from the Danfoss project [65] will be discussed and compared to four aspects of related work: CP-nets and state space methods in industrial settings, temporal logics and model checking in verification tools, the symmetry method and model checking, and finally, topology independence and state space verification.

CP-nets and State Space Methods in Industrial Settings. CP-nets and state spaces have been applied in a number of projects. In the following the use of state spaces and the conclusions from the Danfoss project will be compared to other projects where state spaces have been applied in industrial settings.

- In [54] CP-nets is used for the modelling and analysis of a protocol for remote object invocation in the object oriented programming language BETA [68]. The structure of the modelled BETA system has similarities to the modelled flowmeter system in [65]. In [54] the CPN model models a number of user threads located in shells which are distributed among a number of ensembles whereas the flowmeter system contains a number of CANAPPs distributed among a number of modules. State spaces are used for the analysis of the BETA system. Due to the state explosion problem it was only possible to verify small configurations (up to three user threads) of the system. Net reductions are suggested to alleviate the state explosion problem but did no lead to analysis of larger configurations of the BETA system.

- [77] reports on an industrial cooperation project where CP-nets are used to design the software of a Dalcotech security system. State spaces are used to investigate dynamic properties of the CPN model. Due to the state explosion problem a number of simplifications are done in order to be able to analyse the CPN model of the security system. State spaces are constructed for reduced scenarios and small configurations of the system. According to [77] the state space analysis lead to detection of approximately 15 non-trivial errors. No attempts are made to alleviate the state explosion problem.

- [94] reports on an industrial project where CP-nets are used for the modelling of a software architecture in Nokia mobile phones. The dynamic properties of the CPN model are investigated using state spaces. Due to the state explosion problem the state space cannot be constructed for the full CPN model. Instead state spaces are constructed and analysed for important sub-models. No attempts are made to alleviate the state explosion problem. Instead it is envisioned that it will be possible to construct the state space of the full CPN model on another architecture that the CPN model is in the process of being moved to.

- [18] reports on an industrial cooperation project where CP-nets and state spaces are used for the modelling and analysis of a communication protocol in the Bang&Olufsen BeoLink system. The CPN model is a timed model [47] and, thus, has a infinite state space. Branching options of the Design/CPN OG tool [17] are exploited to obtain finite partial state spaces of the initialisation phase of the communication protocol. In order to make the CPN model suitable for state space analysis [18] introduces bounds on the buffers and considers only small configurations of the BeoLink system.

What is common to the above four projects is that CP-nets and state spaces have been applied in industrial settings. In all four projects state spaces are used to verify the basic properties of the CPN model; not to do a total verification of the system. All four projects encounter the state explosion problem and work around the problem by making simplifications of the models and considering small configurations of the respective systems. However, all of the projects states that state spaces successfully either directly detected errors in the systems or resulted in an increased reliability of the systems. These observations are consistent with the conclusions of the second phase of the Danfoss project [65].

In the Danfoss project the state space analysis is taken a step further by the application of SSPSs to alleviate the state explosion problem. Not many papers report on the use of reduction techniques for state spaces of CP-nets in industrial settings. [87] report on a project where state spaces and state spaces with general equivalences (OE graphs) [15, 47] are used for the analysis of interworking traders. To my knowledge [65] is the first paper reporting on the practical use of the symmetry method for CP-nets in an industrial setting. The use of symmetries yielded significant reduction in the size of the state space, and hence allowed analysis of configurations of the flowmeter system which could not be handled with full state spaces.

Temporal Logics and Model Checking in Verification Tools. In the Danfoss project the properties are verified from the state space using both proof rules for standard dynamic properties of CP-nets [47] and temporal logics [27].

The proof rules are available in the Design/CPN OG tool as a set of query functions [17] whereas verification of formulae specified in temporal logics is supported by the Design/CPN OG tool library ASK-CTL [20].

Temporal logics is widely used for the specification of properties of systems. Several variants exists, however, the basic idea is to build temporal logic formulae from atomic propositions (relating to concepts of the modelling language), proportional operators, e.g., $\vee$ and $\wedge$, and temporal operators, e.g., $\Box$ (always) and $\Diamond$ (exists). Among the two most widely used variants of temporal logics are *Linear Temporal Logics (LTL)* [92] and *Computational Tree Logics (CTL)* [22] which are used as specification languages in a number of model checking tools: LTL and CTL in PROD [76], LTL in SPIN [84], CTL in SMV [69, 82], CTL in PEP [74], and LTL and CTL in the Cadence SMV Model Checker [83].

LTL and CTL can also be used for the verification of properties of CP-nets. The Design/CPN OG tool library ASK-CTL [20], however, supports the verification of properties specified in the language ASK-CTL [12]. ASK-CTL is a *both* state and action oriented variant of CTL, thus ASK-CTL is an extension of CTL which makes it possible to also express properties about the actions of the CP-net, i.e., the information labelling the edges. One of the strengths of CP-nets compared to other modelling languages is that CP-nets are both state and action oriented. Thus, using a both state and action oriented specification language is potentially very useful when expressing properties of CP-nets. The *absence of attribute corruption* property verified in the Danfoss project is concerned with the bindings of three transitions in the CPN model of the flowmeter system. Hence, the property was specified as an action-based CTL formula in ASK-CTL.

The Symmetry Method and Model Checking. A number of papers combine symmetry and model checking of different temporal logics using different modelling languages and representations of the state space. Below a number of the most prominent approaches are discussed.

In [23] and [28] temporal model checking in the presence of symmetry is investigated. The temporal logic considered is CTL* which contains CTL and LTL as subsets and the underlying model is a finite state system (Kripke Model). It is ensured that the condensed state space obtained preserves the truth value of the CTL* formula verified by using a symmetry group for the reduction which is contained in the intersection of symmetries present in the model and the symmetries present in the formula to be verified. This approach corresponds to the suggested "strengthening" of the symmetry specification in [65] to ensure that the symmetry group used for the reduction leaves the truth value of the atomic state propositions of the *absence of attribute corruption* formula invariant.

[30] and [38] presents work on the use of the symmetry method when model checking under fairness assumptions. The basic idea is to restrict the model checking to fair executions[2] of the system being analysed. [30] considers model checking of CTL* whereas [38] deals with LTL. The work in [30] and [38] is

[2]Executions can be considered fair if they fulfil some fairness requirement, e.g., strong fairness or weak fairness.

motivated by the fact that the approaches presented in [23] and [28] intersect the symmetries of the system with the symmetries of the formula to be verified and, hence, when fairness is incorporated in the formula often fail to obtain any reduction at all.

In [23] the use of the symmetry method in symbolic model checking is considered. In symbolic model checking Binary Decision Diagrams (BDDs) are used to calculate and represent the state space. The complications caused by the use of BDDs are investigated and it is shown that with some symmetry groups (rotations and the group of all permutations) the size of the BDD used to represent the orbit relation, i.e., the equivalence relation expressing when states are symmetric, grows exponentially in the minimum of the number of components and the number of states in the component. The reason BDDs does not work well with the symmetry method is caused by the lack of correlation between the size of the state space and the size of the BDD representing the state space. Hence, a reduction in the size of the state space may result in increased size of the BDD.

Topology Independence and State Space Verification In [65] the *topology independence* property was not completely verified since it required state spaces to be constructed for all possible (infinitely many) configurations of the flowmeter system, i.e., initial markings of the CPN model constructed in the Danfoss project. This recurrent problem of the state space method has been studied in, e.g., [36, 91].

In [36] a combination of state spaces and induction is used to do a complete verification of a CPN model of an arbiter cascade. State spaces are used to prove that a single arbiter possesses some desired properties. This is followed by the use of mathematical induction to prove that this was also the case for arbiter cascades of arbitrary depth.

[91] presents results where the correctness of infinite parameterised families of systems are established using process-algebraic compositional finite state verification techniques. The basic idea is to construct a labelled transition system that represent the behaviour of a single component of the parameterised system. The global behaviour of the system is investigated starting from a subsystem such that it contains a minimum number of components and incrementally adding more components. In each iteration the behaviour of the composed system is compared to the behaviour of the previous system. This procedure is continued until a fix-point is reached, i.e., the behaviour of the system with $n + 1$ components is equivalent to the behaviour of a system with n component (if such a fix point exists, otherwise the method fails).

Chapter 3

Exploting Stabilizers and Parallelism

This chapter discusses the paper *Exploiting Stabilizers and Parallelism in State Space Generation with the Symmetry Method* [66]. Section 3.1 describes the background of the paper and contains an introduction to the results presented in the paper. Section 3.2 gives a summary of the paper and discusses the main results of the paper. Finally, Sect. 3.3 contains a discussion of related work.

3.1 Background

A central issue during generation of the condensed state space is to determine whether two states s_1 and s_2 are symmetric, i.e., whether one of the permutation symmetries given by the symmetry specification maps s_1 to s_2. This problem is also referred to as the *orbit problem*. The orbit problem is known to be at least as hard as the *graph isomorphism problem* [23] for which no polynomial time algorithm is known. This paper deals with a derivation of the orbit problem, referred to as the *constructive orbit problem*, which is concerned with computing a canonical (unique) representative for each equivalence class of states. The constructive orbit problem is at least as hard as the orbit problem.

The presented results concentrate on the constructive orbit problem for markings. The reason is that the problem of computing canonical representatives for equivalence classes of binding elements can be reduced to the problem of computing canonical representatives for equivalence classes of markings by viewing the variables of the transition as places and the values assigned to the variable as a singleton multi-set of tokens. Given a transition t with variables $v_1, v_2, \ldots, v_n$, a binding element of (t, b) can be viewed as a vector of singleton multi-sets $(1`b(v_1), 1`b(v_2), \ldots, 1`b(v_n))$ where $b(v)$ denotes the value assigned to v in the binding b. Since transitions cannot be permuted by permutation symmetries in CP-nets [47], finding a canonical representative for $[(t, b)]$ is the same as finding a canonical representative for b which by the above reduction is the same as finding a canonical representative of a marking.

The symmetries used for the reduction are obtained from permutations of the atomic colours in the CPN model. Let Σ_A denote the set of atomic colour sets of the CPN model. For each atomic colour set in the CPN model, $A \in \Sigma_A$, we define an algebraic group of of permutations Φ_A, i.e., a subgroup of $[A \to A]$.

A symmetry ϕ of the system is a set of permutations of the atomic colour sets of the model, i.e., $\phi = \{\phi_A \in \Phi_A\}_{A \in \Sigma_A}$. In the rest of the thesis we will use the term *permutation symmetry* to denote a set of permutations of the atomic colour sets of a CPN model. A *symmetry specification* of a CP-net is an assignment of algebraic groups of permutations to each of the atomic colour sets of the CP-net and hence determines a group of permutation symmetries.

In the following it is assumed that a CP-net with places $P = \{p_1, p_2, ..., p_n\}$ and a consistent symmetry specification SG is given. Φ_{SG} denotes the group of permutation symmetries given by SG.

3.2 Summary of Paper

An essential aspect of calculating condensed state spaces is when reaching a new state/action x during state space generation to check whether a state/action from the same equivalence class is already included in the condensed state space. In the rest of the thesis $[x]$ will be used to denote the equivalence class of states/actions containing the state/action x. In this paper the check for markings of CP-nets is implemented by computing the canonical (unique) representative for each equivalence class by invoking some function CANONICAL which given a marking M calculates the canonical representative of $[M]$. The check then amounts to transforming the marking into CANONICAL(M) and then check (using ordinary equality) whether the resulting state is already included in the condensed state space. Hence, the paper presents a solution to the constructive orbit problem.

The aim of the paper is to investigate techniques to alleviate the negative impact of the orbit problem during state space generation with the symmetry method. The paper attacks the problem in two ways. Firstly, by presenting algorithms which exploits symmetry groups of stabilizers. Secondly, by presenting a parallel version of the basic algorithm for state space generation with the symmetry method.

The basic idea behind the specification of the canonical representatives of equivalence classes of states in CP-nets is to define a total ordering on the markings of CP-nets. The details can be found in [66] which is contained in full in Chapter 8. Having defined such a total order the smallest element in the equivalence class $[M]$ of a marking M will be denoted by $[M]_{min}$. The brute-force approach for calculation of the canonical representative is simply to apply each permutation symmetry given by the permutation symmetry specification in turn and return the smallest resulting marking. The brute-force approach is simple and easy to implement. However, the approach is inefficient in practice. The main problem with this algorithm is that each time a marking is reached during generation of the condensed state space $|\Phi_{SG}|$ permutation symmetries are applied to the marking. If all colours in each of the atomic colour sets can be permuted arbitrarily then Φ_{SG} determines $\Pi_{S \in \Sigma_A}(|S|!)$ permutation symmetries. Practical experiments in [65] have also shown that when using this approach the growth of $|\Phi_{SG}|$ as a function of the system parameters becomes a serious bottleneck in the analysis of systems in practice.

The first contribution of the paper is to use stabilizers (denoted self symmetries in [47]) to improve the complexity of the brute-force approach. The basic idea behind the use of stabilizers is to observe that for a marking M some of the permutation symmetries have similar effects when applied to M. The set of stabilizers of a marking M (denoted Φ_{SG}^{M}) is the subset of Φ_{SG} mapping M to itself. Formally, $\Phi_{SG}^{M} = \{\phi \in \Phi_{SG} \mid \phi(M) = M\}$. It is easy to see that the set of stabilizers for a marking M forms a subgroup of Φ_{SG}. A *left coset* (in the rest of the thesis just referred to as a coset) of Φ_{SG}^{M} is a set of the form $\phi \circ \Phi_{SG}^{M} = \{\phi \circ \phi' \mid \phi' \in \Phi_{SG}^{M}\}$, where $\phi \in \Phi_{SG}$. The set of cosets form a disjoint partitioning of Φ_{SG}^{M} [1]. From this it follows that two permutation symmetries from the same coset of Φ_{SG}^{M} map M to the same marking. Hence, when calculating the canonical representative of M only one permutation symmetry from each of the cosets of Φ_{SG}^{M} has to be applied to M. It follows from La Grange's theorem [1] that the number of representatives, i.e., the number of permutation symmetries that have to be applied is $|\Phi_{SG}|/|\Phi_{SG}^{M}|$.

From the above it follows that the use of stabilizers allows the number of permutation symmetries applied when canonicalizing a marking M to be reduced from $|\Phi_{SG}|$ to $|\Phi_{SG}|/|\Phi_{SG}^{M}|$. This is, however, obtained at the extra cost of calculating the group of stabilizers each time a marking M is canonicalized. In [51] it is suggested to use the Backtrack method [7] to compute the stabilizers of a marking M. The Backtrack method iteratively calculates the stabilizers of a marking. During calculation, the Backtrack algorithm maintains a subgroup of Φ_{SG}^{M} and exploits that it is only necessary to test one permutation symmetry from each coset of the subgroup currently found. Due to the so-called *faster tester property* [51] of the Backtrack method only few permutation symmetries[1] are applied to markings when calculating stabilizers. Hence, it can be argued that the overhead of using stabilizers in the canonicalization of markings is potentially much smaller than what is gained by reducing the number of permutation symmetries applied. This statement is supported by the experimental results presented in [66].

Another observation made in [47] is that stabilizers also can be used to reduce the number of binding elements which have to be considered. The consistency of the permutation symmetry specification ensures [47] that for any two reachable markings, M and M', all binding elements (t, b) and all permutation symmetries $\phi \in \Phi_{SG}$: $M[(t,b)\rangle M' \Rightarrow \phi(M)[(t,\phi(b))\rangle\phi(M')$. This implies that for all $\phi \in \Phi_{SG}^{M}$ it is the case that $M[(t,b)\rangle M' \Rightarrow M[(t,\phi(b))\rangle\phi(M')$. Hence two binding elements (t, b_1) and (t, b_2) enabled in a marking M and satisfying that $(t, b_1) = (t, \phi(b_2))$ for some stabilizer ϕ of M will lead to symmetric markings. Hence only one of (t, b_1) and (t, b_2) needs to be considered for the construction of the condensed state space. Each binding element considered during construction of the condensed state space results in a calculation of the canonical representative for a marking. Hence, the use of stabilizers can be used to reduce the number of calculations of canonical representatives.

[1]In general the faster tester property says that when a subgroup with size less than $m!$ of the group of all permutations of n elements is searched for a property satisfying all elements in the subgroup, at most $m - 1$ permutations are tested [51].

The second contribution of the paper is to exploit parallelism in the state space generation with the symmetry method. The paper [66] presents an algorithm which distributes the canonicalization of markings to a number of processes and in this way do canonicalization in parallel. The basic idea behind the parallel algorithm is to use a number of *slave processes* for the time-expensive canonicalization of markings and a *master process* for construction of the condensed state space itself based on the canonical markings received from the slave processes. The algorithm is shown in [66] which is contained in full in Chapter 8.

A prototype implementing the algorithms exploiting stabilizers and parallelism has been developed. Based on experimental results obtained using the prototype the algorithms are evaluated. The implementation is based on an integration of Design/CPN OG tool [17,26] which supports state space generation and analysis of CP-nets and the GAP tool [34] which implements efficient representations and manipulations of algebraic groups. The reason for choosing to implement the algorithms using an integration of the Design/CPN OG tool and the GAP tool is that in order to evaluate the algorithms proposed in [66] efficient calculations of stabilizer groups is a central issue. Such an efficient calculated can be obtained using the Backtrack algorithm [51]. An implementing of the Backtrack algorithm in the Design/CPN OG tool requires implementation of both efficient group representations and manipulations; concepts that would require implementation of comprehensive new libraries and data structures in the Design/CPN OG tool. On the other hand such algorithms are already present in the GAP tool and in [51] found to be efficient for calculation of stabilizers of markings. The prototype implementation uses the strengths from each of the two tools: The Design/CPN OG tool implements the state space generation algorithm itself and the storage of the nodes and arcs of the state space. The canonicalization function exploiting stabilizers has been implemented in the GAP tool. The integration of the two tools operates as follows: Each time a new marking marking is generated during the condensed state space construction, it is sent to a GAP process that calculates the canonical representative. The representative marking computed is returned to Design/CPN which continues the condensed state space construction with the canonicalized marking received.

The experimental results obtained show that when exploiting stabilizers in the canonicalization of markings a significant speed-up is obtained in the generation time of the state spaces. The speed-up increases when the number of permutation symmetries allowed by the permutation symmetry specification increases, i.e., when the potential gain from applying $|\Phi_{SG}|/|\Phi_{SG}^{M}|$ instead of $|\Phi_{SG}|$ permutation symmetries when canonicalizing a marking M increases. However, even with the use of stabilizers the complexity of the canonicalization of markings, i.e., the constructive orbit problem, still becomes the bottleneck when system parameters grow. The reason is that even with the use of stabilizers it follows from La Grange's theorem that many (in worst case $|\Phi_{SG}|$ permutation symmetries) have to be applied. This is the case for markings with few stabilizers. Also a new problem arise when using coset representatives for the calculation of the canonical form. Coset representatives form arbitrary sets (opposed to Φ_{SG} which form an algebraic group). The GAP tool has data structures for efficient representation and manipulation of algebraic groups. However, for ordinary sets of permutation symmetries the only possibility is to list the elements before applying them to a marking. Hence, for large sets the memory requirements may become a serious bottleneck for the use of coset representatives for the canonicalization of markings of CP-nets.

When considering the results obtained using the algorithms which exploits stabilizers to reduce the number of binding elements considered the numbers are in this case of course highly model dependent. The experimental results obtained using the examples in [66] show that the use of stabilizers for reducing the number of binding elements considered results in much fewer binding elements processed, i.e., fewer markings canonicalized. This again lead to a reduction in the generation time.

The experimental results obtained using the algorithms exploiting parallelism show that when adding more slaves the generation time decreases. The speed-up[2] obtained is significant in the beginning, becomes gradually smaller until a point is reached at which additional slaves result in only marginal or no reduction in generation time. The reason for this is that the generation is fully parallellised, i.e., adding new slaves will not result in more nodes being handled simultaneously. It is also worth noticing that the speed-up increases when system parameters grow, i.e., the number of permutation symmetries allowed by the permutation symmetry specification grows. The reason for this is twofold. Firstly, the canonicalization becomes more and more expensive since the number of permutation symmetries allowed by the permutation symmetry specification grow exponentially with the system parameters. Secondly, as the system parameters increase each node has more successors nodes, hence, more and more nodes can be handled in parallel. This pattern is also evident from Fig. 3.1 which depicts the correspondence between the generation time of the state space and the number of slaves used for the generation. The three graphs show the correspondence for three values of system parameters (7, 8, and 10) in the CPN model, i.e., increasing size of $|\Phi_{SG}|$.

3.3 Related Work

There are several ways to improve state space generation based on explicit state enumeration. Among these are *state reduction methods*, e.g., the symmetry method, which aims at reducing the number of states and *memory reduction methods* which reduce the amount of memory needed to perform the state space generation. With the use of state and memory reduction methods, run-time often becomes a limiting factor [86]. Thus, parallel processing may become an interesting approach towards improving state space generation. In the following the approach and results presented in [66] will be related and compared to related work within these areas. Four aspects of related work will be discussed: parallellisation of state space generation, state space storage with the symmetry

[2]For n slaves the speed-up is calculated as the generation time with only one slave divided by the generation time with n slaves.

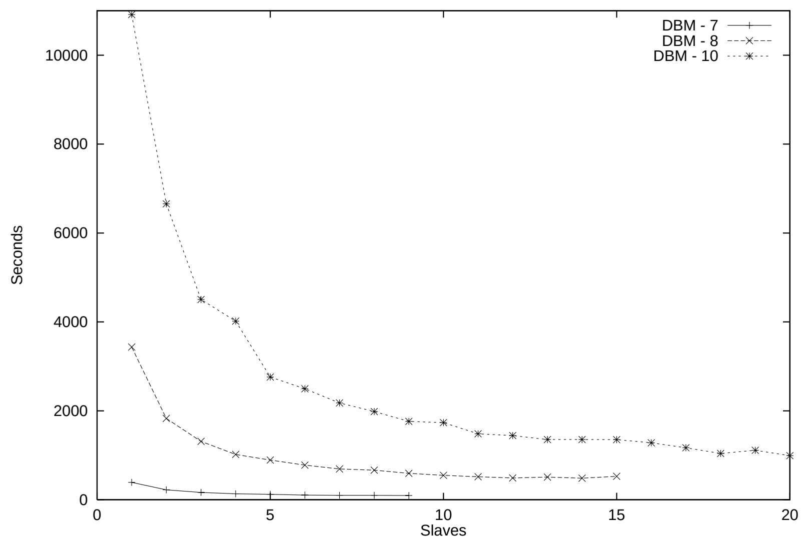


Figure 3.1: Experimental results for system parameters 7,8 and 10.

method, the use of algebraic techniques in state space generation, and finally, canonical forms.

Parallellisation of State Space Generation. The aspect of parallellising the state space generation has also been investigated in the Murφ verifier [86]. In [86] the enabling calculation as well as the storage of the state space are distributed among a number of processes. Each process owns a subset of the states determined by a hash function; when reaching a new state the processes apply a common hash function and send the state to the process determined by the hash value. The parallel Murφ verifier exploits symmetry reduction by applying a canonicalization function before the state is sent to the owning node, whose number is calculated by the hash function. The approach in [66] only parallelise the time expensive canonicalization of states, whereas the enabling calculation and the storage of the condensed state space is located within the master process. The Murφ verifier does not support the checking of temporal properties and according to [86] the parallellisation of verification of temporal properties has not had high priority in the development of the parallel Murφ verifier. One advantage of the technique in [66] is that all standard model checking algorithms can still be used because the state space is stored at a single site.

Recently, the parallellisation of state space storage has been also been investigated in the Maria tool [71]. Based on experimental results in [71] it is concluded that in practice more than 90 % of the time used in state space generation using the Maria tool is used in enabling calculation. The paper presents a simple parallel version of a state space generation algorithm where the enabling calculation is distributed among a number of clients communicating with a server using remote procedure invocation. Like the approach presented in [66] the storage of the state space is centrally handled by the server. [71] presents a number of experimental results and concludes consistently to conclusions in [66] that a small branching factor of the model investigated leads to a decrease in utilisation factor of the clients when the number of clients increase. Similar in [66] adding more slaves does not give any significant speed-up. The approach presented in [71] can without problems be combined with the approach in [66] and it is envisioned that parallelising both the enabling test as well as the canonicalization of states will improve the generation time of the condensed state spaces compared to the results obtained in [66]. The combination of the two approaches will require multiple instances of the Design/CPN tool to run in parallel, i.e., a server responsible for the state space storage and a number of clients responsible for the enabling calculation. This setup can be implemented using the same ideas as in the implementation of the prototype, i.e., Design/CPN running as master process with a number of GAP processes as slaves.

State Space Storage with the Symmetry Method. In the paper summarised in this chapter the condensed state space is generated using canonical representatives, i.e., when reaching a state s a (unique) representative for $[s]$ is calculated and it is checked (using ordinary equality) whether the state is already included in the condensed state space. In the context of CP-nets this idea is new. Traditionally, the first state reached from an equivalence class is the state chosen to represent the equivalence class [47]. When reaching new states it is then checked whether a symmetric state is already included in the condensed state space. However, the idea of using canonical forms is attractive since one of the consequences is that new forms of state storage, e.g., deterministic finite automata (DFAs) [40], binary decision diagrams (BDDs) [93], and graph encoded tuple sets (GETSs) [37] can now be used together with the symmetry method of CP-nets. DFAs, BDDs, or GETSs have not yet been investigated for storing the state space of CP-nets, and a further study is necessary to determine whether such storage techniques combined with canonicalization of states can improve the symmetry method of CP-nets. As already discussed in Chapter 2 the combination of BDDs and the symmetry method may turn out not to be a benefit. The reason is the lack of correspondence between the size of the BDD and the size of the state space that it represents. Thus, reducing the size of the state space using e.g. the symmetry method may not lead to a smaller BDD. Another problem that is common to all three techniques is that it is not in general possible to predict how many bits are needed to store a marking of a place.

Use of Algebraic Techniques in State Space Generation. The use of stabilizers to detect symmetries with similar behaviour is introduced in [47]. In [47] stabilizers are denoted self-symmetries. However, no algorithms nor experimental results are presented. [51] presents an algorithm for computing the set of stabilizers for a marking and the Backtrack method is suggested for this purpose. The results in [51] is based on a manual encoding of the states and did not consider computation of condensed state spaces. The work presented in [66] integrates the use of stabilizers into generation of condensed state spaces using a canonical form calculated from coset representatives of the stabilizer groups. Another contribution of the paper [66] is the integration between the Design/CPN tool and external processes, i.e., instances of the GAP tool. Recently, a new library, Comms/CPN [33], has been developed which extends Design/CPN with the necessary infrastructure to allow CPN models to communicate with external processes, thus allowing this kind of integration to be implemented more easily.

Canonical Forms. In this paper a canonicalization function is used to represent equivalence classes of symmetric states. The canonical form used is quite expensive to calculate but ensures that a unique representative for each equivalence class is calculated. Another approach is to use less expensive normalisation functions [44], i.e., functions which map a state to one element in a set of symmetric states. Hence, the consequence of using normalisation functions instead of canonicalization functions is that several symmetric states can be included in the state space. In [6] a normalisation function is defined on state vectors. The basic idea is to minimise the state vector up to some index i. The value of i determines the amount of reduction between the two extremes: no reduction ($i = 0$) and full reduction (i equal to the size of the state vector). The marking M of a CP-net with places $\{p_1, p_2, ..., p_n\}$ can be represented as a the state vector $(M(p_1), M(p_2), ..., M(p_n))$. Hence, the normalisation function presented in [6] can be applied to CP-nets without major modifications. The use of normalisation is interesting since many of the properties often verified for CP-nets, e.g., boundedness and liveness, do not depend on a perfect reduction.

In [81] a canonical form is specified for P/T-nets. However, the canonical form is restricted to symmetry groups based on all permutations or rotations. For other symmetry groups the canonical form fails and the method does not obtain full reduction. Hence, for some symmetry groups the canonicalization function is instead a normalisation function.

Chapter 4

Algorithms and Tool Support for the Symmetry Method

This chapter discusses the paper *Coloured Petri Nets and State Space Generation with the Symmetry Method* [63]. Section 4.1 describes the background of the paper and contains an introduction to the results presented in the paper. Section 4.2 gives a summary of the paper and discusses the main results of the paper. Finally, Sect. 4.3 contains a discussion of related work.

4.1 Background

The practical use of the symmetry method in the Danfoss project [65] identified two aspects that may improve the practical applicability of the symmetry method for CP-nets: 1) development of techniques to alleviate the negative impact of $|\Phi_{SG}|$ in the orbit problem, and 2) better tool support that automates the symmetry method, and hence does not require the user to possess detailed knowledge about the symmetry method or do a manual implementation of the algorithms involved. This is especially important if the algorithms used for the symmetry reduction are required to be implemented efficiently.

The work presented in this chapter addresses the two aspects through the construction of a tool (the Design/CPN OPS tool [64]) which automates the use of the symmetry method for CP-nets. The Design/CPN OPS tool automatically generates the predicates needed by the Design/CPN OE/OS tool [52], i.e., the two predicates expressing whether two markings and binding elements are symmetric. Since symmetry between binding elements can be seen as a special case of symmetry between markings (see, e.g., Chapter 3) only symmetry between markings are discussed in this chapter.

During development of the Design/CPN OPS tool a number of techniques are developed and refined in order to obtain an efficient symmetry check in practice. The techniques and algorithms are presented in the context of CP-nets. The results are, however, also valid for other modelling formalisms where the symmetry method is applicable.

Symmetry in a CP-net is specified by means of a *symmetry specification* which assigns an algebraic group of permutations Φ_A to each atomic colour set A in the CP-net. A *permutation symmetry* ϕ in the CP-net is a set of permutations on the atomic colour sets of the CP-net, i.e., $\phi = \{\phi_A \in \Phi_A\}_{A \in \Sigma_A}$. A symmetry specification SG determines an algebraic group Φ_{SG} of permutation symmetries. A symmetry specification for a CP-net is required to be consistent, i.e., to determine symmetries which are present in the CP-net. Two markings M_1 and M_2 are symmetric if $\exists \phi \in \Phi_{SG} : \phi(M_1) = M_2$, and similar for binding elements.

A data structure called a *restriction set* [4,47] can be used to represent sets of permutations of an atomic colour set. Below restriction sets are introduced by means of an example. Let A= {a1,a2,a3} be an atomic colour set. We use a restriction set R to represent a subset of [A $\rightarrow$ A].

$$\mathrm{R}= \begin{array}{|c|c|} \hline \texttt{a1} & \texttt{a2} \\ \texttt{a2 a3} & \texttt{a1 a3} \\ \hline \end{array}$$

Each row in the restriction set represents a requirement that a permutation must fulfil in order to belong to the set of permutations represented by the restriction set. The individual rows express that the colours in the left-hand side must be mapped into the colours of the right-hand side. Hence, the set of permutations of A represented by R is the set of permutations where a1 is mapped to a2 and the set {a2,a3} is mapped to the set {a1,a3}. Arbitrary sets of permutations can be represented as a set of restriction sets [4]. A symmetry specification for a CP-net can be represented by a set of restriction sets for each atomic colour set of the CP-net.

In the following it is assumed that a CP-net with places $P = \{p_1, p_2, ..., p_n\}$ and a consistent symmetry specification SG is given. Φ_{SG} denotes the group of permutation symmetries given by SG.

4.2 Summary of Paper

The paper presents techniques for determining whether two markings of a CP-net are symmetric, i.e., given M_1,$M_2 \in \mathbb{M}$ determine whether $\exists \phi \in \Phi_{SG}$ such that $\phi(M_1) = M_2$. The techniques are implemented in the Design/CPN OPS tool and evaluated in practice. Two CPN models are used for the evaluation: a CPN model of a two-phase commit protocol with w symmetrical workers [53] and a CPN model of a distributed database with n symmetrical database managers [46]. Common for the two CPN models is that $|\Phi_{SG}|$ grows exponentially when system parameters increase. Since the aim of the paper is to develop techniques which can handle large symmetry groups the two models are suitable for the evaluation.

Brute-force Approach. First the paper presents a brute-force approach which given two markings lists and checks all permutation symmetries in Φ_{SG}. An algorithm P_M^{Basic} implementing the brute-force approach is included in the Design/CPN OPS tool and evaluated using the two CPN models of the two-phase commit protocol and the distributed database. The brute-force approach is exactly the approach which fails when calculating canonical representatives of markings in [66]. Hence, the approach is included in this paper for reference purposes.

Consistently with [66] the experimental results show that when using P_M^{Basic} to determine symmetry between markings run-time increases significantly when system parameters grow. Furthermore, the experimental results show that the memory use becomes a serious bottleneck when using large symmetry groups for the reduction. The reason is that the entire Φ_{SG} is listed and kept in memory during the symmetry check.

The paper presents two improved algorithms; one improving the run-time compared to the brute-force approach and one improving the memory use incurred by the listing of permutation symmetries.

Improving Run-time. The basic idea is to narrow the set of permutation symmetries that need to be checked. When checking whether two markings M_1 and M_2 are symmetric a set Ψ_{M_1,M_2} is calculated such that $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\} \subseteq \Psi_{M_1,M_2} \subseteq \Phi_{SG}$. The applicability of the approach depends on how close Ψ_{M_1,M_2} is to $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$ and how efficient Ψ_{M_1,M_2} can be calculated.

In [4,47] it is shown that if a CP-net only contains places with atomic colour sets then the set $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$ of permutation symmetries mapping M_1 to M_2 can be determined efficiently using operations on restriction sets. The basic idea is to look at each place p in isolation and construct a restriction set with restrictions expressing that the set of colours appearing with coefficient i in $M_1(p)$ must be mapped into the set of colours appearing with coefficient i in $M_2(p)$. The restriction set contains one restriction for each coefficient appearing in $M_1(p)$ and $M_2(p)$. For each atomic colour set the restriction sets constructed for places with that colour set are intersected. The resulting restriction sets, i.e., one for each atomic colour set of the CP-net, represent (when intersected with Φ_{SG}) the set of permutation symmetries between M_1 and M_2. Since restriction sets can be efficiently intersected [4] the cost of the above calculation is very low compared to brute-force approach.

Unfortunately, the above approach only works for CP-nets without places with structured colour sets. Nevertheless, we will use the idea to obtain more efficient symmetry checks between markings of arbitrary CP-nets. The idea is to perform the above calculation for the places of the CP-net with atomic colour sets to obtain a set Ψ_{M_1,M_2} of permutation symmetries. Ψ_{M_1,M_2} is a super-set of $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$, i.e., $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\} \subseteq \Psi_{M_1,M_2} \subseteq \Phi_{SG}$. Hence, the original goal of obtaining an approximation of the permutation symmetries is fulfilled and the calculation can be done efficient since the calculation only rely on intersections of restriction sets.

Ψ_{M_1,M_2} is only an approximation of the actual set of permutation symmetries mapping M_1 to M_2. This means that the permutation symmetries in Ψ_{M_1,M_2} have to be checked like the brute-force approach. However, it can be noted that it is only necessary to check the permutation symmetries in Ψ_{M_1,M_2} on the markings of the places with structured colour sets; the places with atomic colour sets are already accounted for in the approximation.

An algorithm P_M^{Approx} for checking symmetry between two markings based on the approximation technique is included in the Design/CPN OPS tool and evaluated using the same two CPN models as for the P_M^{Basic} approach.

How much the approach will improve the run-time of the symmetry check between two markings compared to the brute-force approach is highly dependent on how close Ψ_{M_1,M_2} is to the actual set of permutation symmetries mapping M_1 to M_2. This measure is specific to the CP-net for which the SSPS is generated and depends on the amount of redundancy encoded in the structured colour sets. The experimental results obtained using the P_M^{Approx} algorithm show that for the CPN models of the two-phase commit protocol and the distributed database only a single permutation symmetry is tested after the approximation. This is in practice often the case if the approximation is close or equal to the actual set of permutation symmetries. However, the experimental results also show that even though the approximation dramatically reduces the number of permutation symmetries that have to be checked, it is not possible to generate SSPSs for large Φ_{SG}. The reason is that the approach has the same bottleneck as the brute-force approach caused by the memory required to list the set of permutation symmetries which have to be checked. This problem appear when symmetry is checked between two markings M_1 and M_2 for which $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$ (and hence also Ψ_{M,M_2}) is large.

Improving Memory Use. The goal for the algorithm is to alleviate the problems experienced with P_M^{Basic} and P_M^{Approx} when checking a large set of permutation symmetries. The basic idea behind the algorithm is to maintain a compact representation of the set of permutation symmetries that are checked.

A set of permutation symmetries represented by a set of restriction sets (one for each of the atomic colour sets in the CP-net) can be viewed as a tree: Each atomic colour is assigned a layer in the tree. The nodes in that layer correspond to possible images of the colour. The possible images of a colour c are found from the right-hand side of the restriction containing c in the left-hand side. When reaching a leaf in the tree each atomic colour in the CP-net is mapped into another colour and the corresponding permutation symmetry can be found following the path from the root to the leaf. Figure 4.1 shows a tree representing Φ_{SG} for the distributed database. The CP-nets contain two atomic colour sets `DBM`= {`d(1)`,`d(2)`,`d(3)`} and `E`= {`e`}. The symmetry specification SG allows all permutations of the colours in `DBM`. Hence, Φ_{SG} contains 6 permutation symmetries. Below each leaf the corresponding permutation symmetry is represented as a restriction set.

The basic idea is to avoid listing the set of permutation symmetries. Instead the restriction sets are unfolded as shown in Fig. 4.1 using a number of depth first recursive calls. Each node in the tree corresponds to a recursive call. When reaching a leaf the corresponding permutation symmetry ϕ is checked. If $\phi(M_1) = M_2$ the algorithm stops. Otherwise, the algorithm backtracks until an unexplored path is found. In this way only one permutation symmetry is kept in memory at a time.

An algorithm $P_M^{Approx+Lazy}$ implementing the symmetry check between mark-

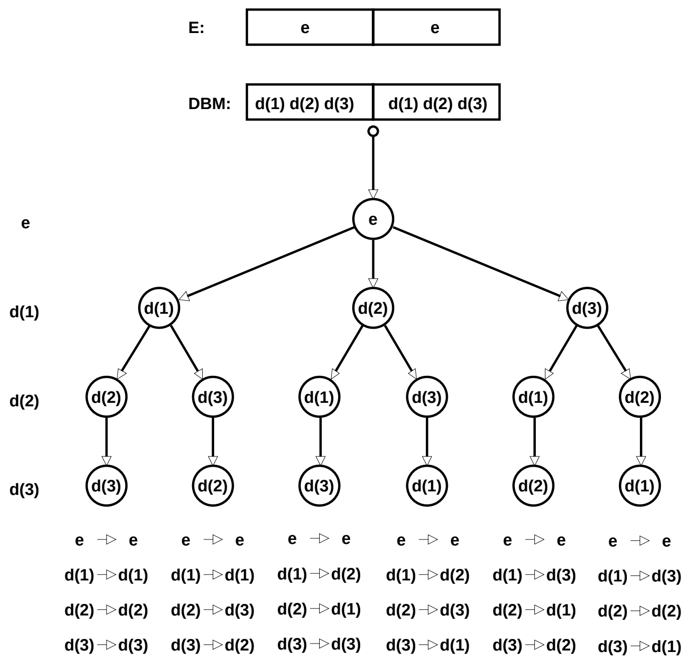


Figure 4.1: All permutation symmetries in Φ_{SG} represented as a tree.

ings using both the approximation technique and recursive unfoldings of the restriction sets has been included in the Design/CPN OPS tool. The algorithm is evaluated using the CP-nets modelling the two-phase commit protocol and the distributed data base. The experimental results show that the bottleneck caused by the memory used to list large sets of permutation symmetries is avoided; using $P_M^{Approx+Lazy}$ SSPSs are generated without problems for CP-nets with $|\Phi_{SG}| = 1.3 \cdot 10^{12}$. It should be noted that the difference between P_M^{Approx} and $P_M^{Approx+Lazy}$ only is the way sets of permutation symmetries are represented while being checked. Since $P_M^{Approx+Lazy}$ also rely on approximation the number of permutation symmetries that are checked is the same.

4.3 Related Work

The main contribution of the work presented in [63] and summarised in this chapter is the development of the Design/CPN OPS tool which together with the Design/CPN OE/OS tool automates SSPS generation for CP-nets with consistent symmetry specifications. Hence, the Design/CPN OPS tool relies on a user specified symmetry specification. A major part of the work concerning the development of the Design/CPN OPS tool is also development and evaluation of techniques to obtain efficient generation of SSPSs in the context of CP-nets; both in run-time and memory use. In the following the work is related and compared to three aspects of related work: the symmetry method in other formalisms, symmetry in Place/Transition-nets, and backtrack methods for exploration of permutation groups.

The Symmetry Method in Other Formalisms. Symmetries in CP-nets were originally introduced in [41] and further developed in [42, 47, 48]. The symmetry method as a state space reduction method is however not restricted to CP-nets.

Well-Formed Nets (WFNs) [14] is a class of high-level nets. Compared to CP-nets WFNs have more restricted and structured colour domains and operations. The restrictions are introduced to ensure that symmetries in the state space can be automatically determined from the colour definitions and inscriptions in the WFN.

In the Murφ verifier [44] a data type called a *scalar set* is introduced to describe symmetries in the system. The operations on scalar sets are restricted to ensure that states have the same future behaviour up to permutations of the elements of the scalar sets. The symmetry reduced state space is automatically generated. A similar approach is taken in Symmetric Spin [6].

[85] introduces the symmetry method in the context of Place/Transition-nets (P/T-nets). In P/T-nets the symmetries considered are permutations of places and transitions. The efficiency of the symmetry method for P/T-nets is studied in [80,81] which will be discussed later in this section. For P/T-nets the symmetry method is fully automatic and in contrast to CP-nets the symmetries are automatically detected.

In theory a CP-net can be unfolded to an equivalent P/T-net [46], i.e., a P/T-net with the same behaviour, and the techniques developed for the symmetry method in the context of P/T-nets can be applied to the unfolded net. However, in practice this approach is not suitable since the unfolded net usually is to large (or even infinite) to be handled in practice.

Symmetry in Place/Transition-nets. In [85] it is shown how symmetry in a P/T-net describes symmetry in the state space. Hence, symmetry in P/T-nets can be expressed as permutations of places and transitions. [81] presents techniques to obtain efficient generation of the SSPS for P/T-nets. Consistently with the conclusions in [63] of the P_M^{Basic} algorithm [81] finds that an approach based on listing and checking the entire group of permutations when checking whether two states are symmetric is not applicable in practice. A compact representation of the symmetry group is obtained using a data structure called a generating set.

A solution is presented that narrows the number of symmetries that have to be checked, i.e., an approach similar to the approximation technique summarised in this chapter. The approximation technique used in [81] is based on the observation that when testing whether two markings M_1 and M_2 are symmetric only permutations for which $M_1(p) = M_2(\phi(p))$ for places p have to be tested, i.e., only permutations of places containing the same number of tokens in the two markings.

Backtrack Methods for Exploration of Permutation Groups. Let S_n denote the group of all permutations of the n elements in a set S. In [2] an algorithm for generating coset representatives of a subgroup G of S_n is presented.

Similar to the approach used in $P_M^{Approx+Lazy}$ (the LazyList algorithm in [63]) the algorithm assigns images to the elements of G in a number of recursive calls. The goal of the algorithm is to produce coset representatives of subgroup G in S_n , hence, only one image from each orbit of G is considered. In each recursive call G is re-calculated as the subgroup which stabilizes all previous choices of images to elements in S. The algorithm proceeds in a number of recursive calls each assigning an image to an element from S until the paths corresponding to permutations that represents the cosets have been explored. In that sense the algorithm is very similar to the LazyList algorithm. The difference is that LazyList explore all paths corresponding to all elements of a subset of S_n whereas the algorithm presented in [2] explore a subset of S_n corresponding to coset representatives of a subgroup of S_n. When combined with the ideas from the LazyList algorithm the algorithm presented in [2] can be used to obtain a solution to the problem experienced in the use of stabilizers in [66] and summarised in Chapter 3. The solution avoids listing all coset representatives of the stabilizer groups used when canonicalizing a marking M: each time a leaf is reached, i.e., a coset representative ϕ is generated, the coset representative is applied to M. During calculation only the smallest marking seen so far has to be remembered and the permutation symmetries can be thrown away when they have been checked. Hence, the solution possess the same advantages as LazyList, i.e., the memory requirements decreases since all coset representatives do not have to be kept in memory at the same time.

Chapter 5

Modelling and Analysis of Feature Interactions

This chapter discusses the paper *Modelling of Features and Feature Interactions in Nokia Mobile Phones using Coloured Petri Nets* [67]. Section 5.1 describes the background of the paper and contains an introduction to the results presented in the paper. Section 5.2 gives a summary of the paper and discusses the main results of the paper. Finally, Sect. 5.3 contains a discussion of related work.

5.1 Background

The paper *Modelling of Features and Feature Interactions in Nokia Mobile Phones using Coloured Petri Nets* [67] presents an industrial cooperation project where CP-nets and its supporting Design/CPN tool are applied for the modelling and initial analysis of feature interactions in Nokia mobile phones. The project is a joint project between Nokia Research Centre, Finland and the CPN group at the University of Aarhus, Denmark. In the rest of this chapter the project is referred to as the MAFIA[1] project.

Nokia mobile phones provides an increasingly diverse set of *features*. Besides basic communication facilities there is a growing demand for entertainment, personal management, etc. During operation of the mobile phone many features are potentially active at the same time. In Nokia much research is devoted to analyse which user tasks are more frequent and support smooth and flexible operation of these tasks, e.g., by providing special keys or short cuts via the user interface (UI) of the mobile phone. Hence, Nokia mobile phones supports the coexistence and interplay of many features. Such dependencies or interplay of features are called *feature interactions* and range from simple usage dependencies to more complex combinations of several independent features. Features are also called applications and in the rest of this chapter as well as in [67] the two terms will be used interchangeably.

The fact the many different kinds of feature interactions are allowed, and

[1]MAFIA is an acronym for Modelling and Analysis of Feature Interactions in mobile phone Architectures.

even supported through the UI of the mobile phone, means that the use of the mobile phone appears both flexible and powerful to the user. On the other hand the feature interactions have shown to introduce an increased complexity as well as problems in the design and development of software in Nokia mobile phones. The motivation behind the MAFIA project was to investigate the feature interactions occurring in the Nokia mobile phones. Before the project started problems were often realised during implementation and integration of individually developed features. By the development of a systematic methodology to describe and document feature interactions in the software design process it was envisioned that the problems leading to costly delays in the integration phase of features could be minimised.

5.2 Summary of Paper

One of the main activities in the project was the construction of a CPN model modelling the software system in Nokia mobile phones[2]. This activity was the main activity for the author of this thesis.

Before the CPN model was constructed the project group collected information and internal documentation about the software system, features and feature interactions in the mobile phones. Furthermore, a group of involved users[3] were established with the purpose of formulating the needs and intentions of the project as well as performing ongoing evaluations of the constructed CPN model during the project. Hence, the CPN model was constructed in close cooperation with the future users of the results produced by the MAFIA project.

In the following the three activities of the project related to the construction and evaluation of the CPN model are summarised: modelling, simulation and analysis.

Modelling. The CPN model is constructed as a hierarchically CP-net [46] in the Design/CPN tool [26]. The CPN model does not model the full mobile phone software system but concentrates on a number of selected features which the project group (including the users) find interesting seen from a feature interaction perspective, e.g., features which are known to cause problems when integrating with new features.

The basic components in the mobile phone software system and the communication protocol is modelled to serve as a framework for the features. Since, the main focus in the MAFIA project is on features and feature interactions it has been the intention to keep the framework as simple as possible and instead focus on the modelling of features. The page shown in Fig. 5.1 is the top-most page of the CPN model, thus, modelling the most abstract view of the mobile phone software system. UIController, Servers, CommunicationKernel and Applications correspond to four central concepts of the mobile phone software system

[2]The software varies between the different product families. The MAFIA project focused on the software used in the Nokia 6210 product family.

[3]The users are a combination of customers, i.e., initiators of the project, as well as involved groups realising problems with feature interactions in their daily work, e.g., software developers, UI designers and testers.

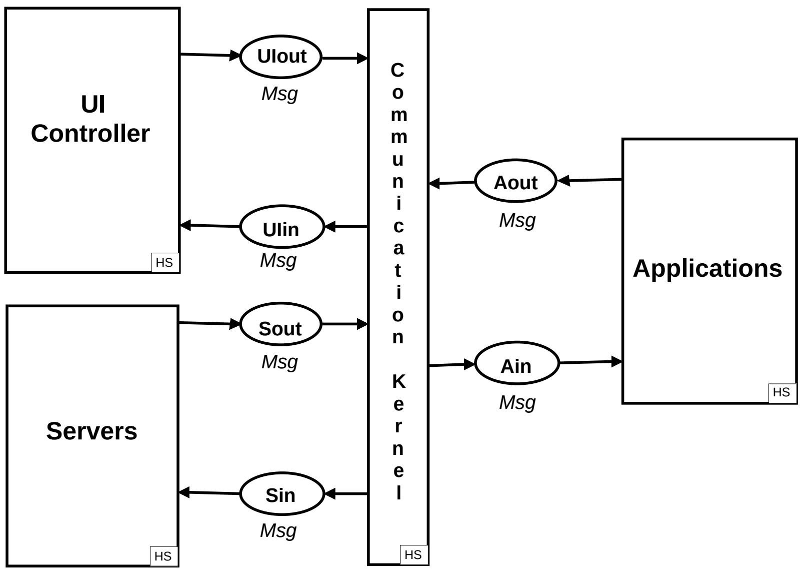


Figure 5.1: The top-most page of the CPN model.

and are all modelled as substitution transitions, which means that their detailed behaviour are modelled on the subpages with the corresponding names. The selected features are modelled and included in the framework introduced above.

The page Applications, i.e., the subpage of the Applications substitution transition in Fig. 5.1, modelling the individual features is shown in Fig. 5.2. Each feature is modelled separately and communicate with the rest of the mobile phone software and other features through the message buffers AOut and AIn. Hence, the modularity of features is exploited using modules (in CPN terminology referred to as pages) in the CPN model. For a more detailed presentation of the CPN model see [67] which is contained in full in Chapter 10.

Via the construction of the CPN model the project group gained valuable insight in the features and their behaviour in the mobile phone software system. One of the results was the identification of similarities in the communication patterns in which the different features was engaged. The similarities were exploited via reuse of subpages modelling the basic communication in the CPN model. The amount of reuse can be illustrated via some statistics from the constructed CPN model: the CPN model contains 25 pages but 110 page instances, i.e., the page modelling the general communication pattern is reused 85 times, thus exploiting the similar basic behaviours of features.

Simulations. During construction the CPN model was validated using simulations; both interactive (step-by-step) and more automatic simulations. These kind of simulations where the token game is observed during and after simulations of the CPN model were extremely useful for the members of the project group who were experienced in using CP-nets and the Design/CPN tool and who had a detailed knowledge of the CPN model constructed in the MAFIA project. However, the close cooperation with the users without any previous

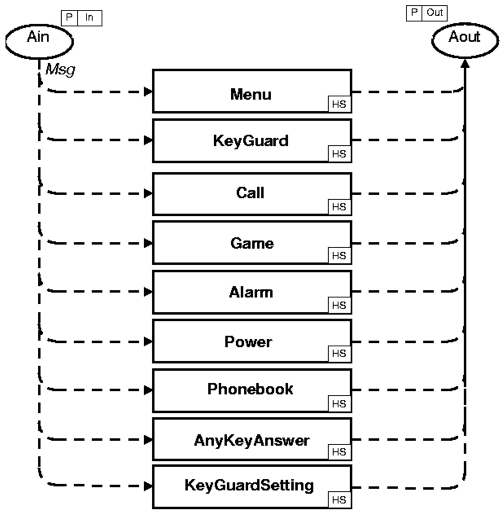


Figure 5.2: The page Applications modelling the individual features included in the CPN model.

knowledge of modelling techniques or in particular Petri Nets made it at an early stage in the project clear that some kind of visualisation was necessary to supplement (or even replace) the observation of the token game during simulations.

A central issue of the CPN model constructed in the MAFIA project is that it is intended to be used by different groups of users; from UI designers who are interested in the behaviour of features and their interactions at the level of the UI of the mobile phone to software developers who are interested in details about the communication between features involved in a feature interaction. Hence, one of the main goals in the project was to be able to reflect these different levels of abstraction in the visualisation techniques used for the CPN model. Two extensions have been made to the CPN model to visualise and control simulations: domain specific graphics and Message Sequence Charts [45]. Each of them are summarised below.

Domain Specific Graphics. The CPN model is extended with domain specific graphics using the Mimic library [78] of Design/CPN. The Mimic library allows the CPN model to be extended with graphical objects which can be displayed, hidden or updated during simulations of the CPN model. Furthermore, using the Mimic library user input, such as mouse clicks on graphical objects, can be read into the CPN model during simulations. Using the Mimic library the CPN model is extended with facilities for both controlling simulations and visualising the state during simulations using domain specific graphics.

The CPN model is extended with a picture of a mobile phone animating the display during simulations. Figure 5.3 shows a snapshot of the animation taken during a simulation of the CPN model. The snapshot shown corresponds to a state of the CPN model where the mobile phone indicates an incoming

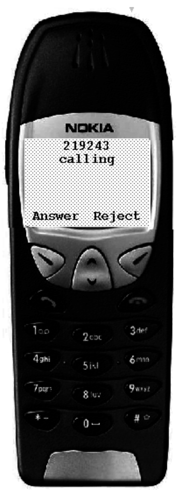


Figure 5.3: Domain specific graphics animating the display during simulations.

call. The animation is constructed as a number of layered graphical objects: 1) a background picture of a Nokia mobile phone as it is known to the users of the CPN model, and 2) on top of the display of the background picture there are a number of graphical objects. Updating the contents of those graphical objects allow dynamic animation of the display of the mobile phone during simulation. Functions for updating the graphical objects are called from code segments [46] of the CPN model.

The picture of the mobile phone is also used to control the simulation. By means of key presses (using mouse clicks), the user can control the simulation, e.g., answering the incoming call or start a new game. On top of each of the keys of the background picture there is a graphical object. Using the facilities provided by the Mimic library key presses on such objects are read into the CPN model and used to control the simulations. Additional graphical objects are used in the model for other kinds of visualisations and possibilities to control the simulations. Details can be found in [67] which is contained in full in Chapter 10.

Message Sequence Charts. The domain specific graphics visualise the behaviour of the CPN model at the level of the UI of the mobile phone. The animation of the display dynamically reflects the contents of the display as the user of the mobile phone observes it. However, the individual features and their communication in the system are not reflected; aspects that are central to, e.g., software developers. Therefore, the visualisation facilities of the CPN model developed in the MAFIA project was extended with Message Sequence Charts[4] (MSCs) [45] which capture the communication between features, servers and the UI controller in the mobile phone software system. To support the need for different abstraction levels in different kind of users' use of the CPN model the

[4]Message Sequence Charts have been presented in the discussion of the Danfoss project in Chapter 2 and will therefore not be further introduced here.

MSCs can be dynamically turned on and off during simulations. Also the level of detail in the MSCs can be changed dynamically, i.e., it is possible to abstract away different kinds of communication as well as features.

During the MAFIA project the MSCs were heavily used in the produced documentation about feature interactions in Nokia. The animation of the display, on the other hand, does not capture information for later use. To overcome this problem the MSCs are extended by adding information about the appearance of the display. Each time the display is updated in the animation a new picture is added in the MSC. In this way the correspondence between the communication in the system and the contents of the display is captured for later use or documentation. Figure 5.4 shows an example of an extended MSC. From the figure it can be seen that the MSC capture information about both the communication and the contents of the display as well as the interrelation between the communication and changes in the appearance of the display.

Analysis. Formal analysis of the CPN model was not a part of the MAFIA project. However, initial analysis has been made to evaluate the applicability of the modelling techniques and chosen abstraction level for features to automatically analyse features and detect feature interactions from the CPN model. The main goal of the initial analysis was to evaluate whether the CPN model is suited for analysis in preparation for future projects within Nokia.

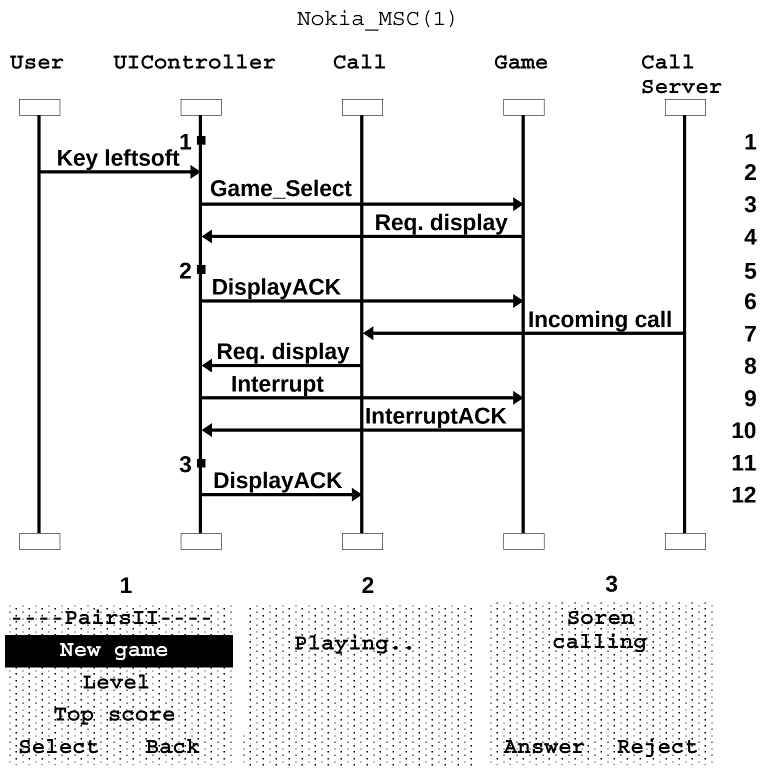


Figure 5.4: An example of the use of MSCs.

The CPN model was analysed using full state spaces and the Design/CPN tool [17]. The state space for the full CPN model, i.e., the framework and the all of the features included in the CPN model, is expected to be very large. An effort has been made to construct the CPN model in such a way that features can be removed and added without having to change the rest of the CPN model, i.e., the framework. This fact was exploited in the analysis of the CPN model; most of the features were removed and the analysis started with simple configurations of the mobile phone software system to get an idea of the sizes of the state spaces involved.

One of the aims of the analysis was to automatically detect feature interactions from the state spaces of different configurations of the CPN model. Hence, feature interactions need to be formally specified in terms of state spaces. In the following $\mathcal{S}_{f_1,f_2}$ denotes the full state space of the basic CPN model including the features f_1 and f_2. $\mathcal{S}_f \models P$ means that the property P can be verified from $\mathcal{S}_f$. One possible way of formally expressing that two features f_1 and f_2 interact is that $\mathcal{S}_{f_1} \models P$ but $\mathcal{S}_{f_1,f_2} \not\models P$ for a property P.

This way of specifying feature interactions will be illustrated using an example: a property P of the Any key answer feature can be used to automatically detect a feature interaction from the state spaces of the CPN model in different configurations. Informally, P is the property that *"if a call comes in and the any key answer setting is on then pressing any key of the mobile phone will answer the call"*. Using the temporal logics ASK-CTL [12, 20] P is formulated as a query in the Design/CPN OG tool and the answer can be automatically determined when a state space has been generated. When verifying P in different configurations of the CPN model it is found that $\mathcal{S}_{\mathsf{AnyKeyAnswer}} \models P$ but $\mathcal{S}_{\mathsf{AnyKeyAnswer,KeyGuard}} \not\models P$. Hence, a feature interaction is detected between the Any key answer feature and the Key guard feature.

Using the approach presented above a number of feature interactions are detected from the CPN model in smaller configurations (up to three features). After having performed this initial analysis it is concluded that the CPN model developed in the MAFIA project seems applicable for analysis and detection of feature interactions; no major changes or adjustments are needed. It is, however, envisioned that the analysis may benefit from experimenting with different specifications of feature interactions in terms of state spaces. The approach used for the initial analysis is well suited for the detection of a certain kind of feature interactions, i.e., feature interactions where the behaviour of a feature is modified by the presence of another feature. Other categories of feature interactions, e.g., simple usage dependencies, may be more easily detected using other strategies.

As a final remark it should be noted that the state space of the full CPN model is expected to be very large, however, the use of reduction methods has not been investigated. Since the features of the mobile phone are asynchronous a possible direction for future analysis would be to use the stubborn set method [88] to reduce the size of the state space.

5.3 Related Work

The MAFIA project demonstrates the use of CP-nets in an industrial setting. The project deals with features and feature interactions in Nokia mobile phone software; concepts which have also been studied in other application areas. An important aspect of the CPN model developed in the MAFIA project is the use of domain specific graphics for visualisation of the behaviour of the CPN model. In the following the work in the MAFIA project will be discussed and related to work within the above mentioned areas, i.e., the feature interaction problem and domain specific graphics in industrial settings.

The Feature Interaction Problem. The feature interaction problem has been studied in several application areas, e.g., telecommunication services (see [56] for a survey), process planning [43], and computer-aided design [75]. Especially in the area of telecommunication services much research on the feature interaction problem has been done and there is a biannual workshop on the topic [9, 57].

The area of telecommunication services deals with the services offered to the users of a telecommunication company, e.g., call forwarding and voice mail. Services are composed of a number of features. In the area of telecommunication services the term feature interaction refer to situations where two or more features affect the behaviour of each other. The features may belong to the same or different telecommunication services. The problems related to feature interactions studied in the MAFIA project are more general, i.e., how to describe and coordinate concurrent interrelated processes.

In [8] three major trends in the field of feature interactions in telecommunication services are identified: software engineering approaches, formal methods, and on-line techniques. The MAFIA project can be characterised to belong to the software engineering approach [67]. However, in the following it is discussed how the final results from the MAFIA project relates to aspects within all three approaches:

Software engineering. The software engineering approach is characterised as the use of development models and techniques from software engineering to address the feature interaction problem in the creation of telecommunication services. Formal methods may be a part of the development models and techniques, where they can be used to add rigour to the service creation. However, the use of formal methods to analyse or detect feature does not fall into the software engineering approach [8]. In the MAFIA project CP-nets are used to address the feature interaction problem in Nokia mobile phones. The main goal was to use modelling techniques to improve the specification and development process of features; both by resolving inconsistencies in the specifications as well as developing a systematic methodology for the development of features. The use of an underlying formal method, i.e., CP-nets, ensured that a formal description of features was obtained. At the same time the constructed formal specifications could be used to investigate the dynamical behaviour of features and their interactions.

Formal methods. The formal methods approach is characterised by the use of formal methods to either analyse features or detect feature interactions. In the area of telecommunication services formal methods are often used to analyse interactions at the level of the service specification, i.e., interactions that are independent of the actual implementation [8]. In the MAFIA project formal methods, i.e., state spaces and model checking techniques, were used to perform initial analysis of the features in the CPN model and evaluate the applicability of the CPN model for automatic detection of feature interactions. When using model checking techniques and state spaces the abstraction level of the analysis results obtained is highly dependent of the abstraction level of the underlying model. Since the CPN model constructed in the MAFIA project is a model of the mobile phone software system of a family of Nokia mobile phones, the analysis results produced using the formal methods in the MAFIA project is (in contrast to the trend in telecommunication services) implementation dependent.

On-line techniques. The on-line techniques approach is characterised as techniques applied at run-time. The techniques can be based on detection and/or resolution of feature interactions. An example is the use of a feature manager to detect and resolve feature interactions [8]. During construction of the CPN model in the MAFIA project an increased understanding of the features and feature interactions in the mobile phone software system was obtained and improvements were suggested to the design of mobile phone software system. The suggested improvements include a redesign of the UI controller which handles the features' requests to access the UI of the mobile phone. In the current design of the mobile phone software system features are responsible for resolving potential interactions based on fixed rules for each pair of features. In the suggested new design feature interactions are detected and resolved in the UI controller when the features request access to the UI. Hence, the modelled UI controller detect and resolve feature interactions caused by conflicting UI requirements at run-time and thus work as a feature manager as described in [8].

A majority of the work in the area of feature interactions in telecommunication services, e.g., [10, 73], concentrate on the use of formal methods:

In [10] the SPIN tool [39] is used for the analysis of feature interactions in telecommunication services. The underlying model is constructed as a Promela description and consists of a basic call model and six features that can be added incrementally as in-line Promela functions. This approach with a separation between the basic call model and the features, thus allowing incremental state space analysis, is similar to the modelling approach in the MAFIA project. The CPN model of the mobile phone software system exploits modularity through the use of subpages and features can be added and removed without changing the rest of the CPN model. The Promela description of the telecommunication services is analysed using state spaces and the verified properties are specified in LTL. The feature interactions are in [10] specified the almost the same way[5]

[5]The only difference between the specification of feature interactions in [10] and [67] is due to the fact that features may belong to several users in [10] whereas [67] only considers a single-user system.

as in the MAFIA project, i.e., a feature interaction occur if a property P that holds for a feature f_1, i.e., $\mathcal{S}_{f_1} \models P$, no longer holds in the presence of another feature f_2, i.e., $\mathcal{S}_{f_1,f_2} \not\models P$.

The state space analysis performed in the MAFIA project is only in an initial phase and, hence, no attempts have been made to alleviate the state explosion problem. Since the state explosion problem is a general problem when state spaces are applied for the analysis of systems, methods for alleviating the state explosion problem have, not surprisingly, also been studied in the context of the feature interaction problem.

In [10] partial order methods are used to reduce the sizes of the constructed state spaces; a method that in [67] is expected to work well with the CPN model constructed in the MAFIA project. The method is not applied in the MAFIA project, however, since tool support for partial order methods for CP-nets [60, 61] was not mature enough in the Design/CPN tool to be used in the MAFIA project.

In [73] telecommunication services are modelled using finite state machines. In order to deal with services with many users the symmetry method is used to reduce the size of the state spaces. Users of the services are considered symmetric and in the analysed configurations [73] obtains 80% reduction in generation time and space. The results presented in [73] can not, however, be transferred to the MAFIA project since the CPN model developed in the MAFIA project models a single mobile phone and the features do not possess symmetry.

Domain Specific Graphics in Industrial Settings. An important aspect of the CPN model of the mobile phone software system developed in the MAFIA project is the use of domain specific graphics to visualise and interact with the CPN model. In the following other industrial projects where CP-nets and domain specific graphics have been used are presented and the contributions of the MAFIA project are discussed.

- The Design/CPN Mimic library [78] was originally motivated by the project reported in [77]. In the project CP-nets were used to design a Dalcotech security system, i.e., an intruder alarm system. The system consists of a central unit which handles a number of components, e.g., detectors and indicators. In the project simulations were used to investigate the dynamic properties of the CPN model. The desire for more "user friendly" [77] simulations lead to the development of the Design/CPN Mimic library [78]. Using the library an enhanced user interface was build visualising a house with the modelled intruder alarm system. The user interface supports two way communication between the user and the simulation; during the simulations the current state of the CPN model is visualised, e.g., indicating horns and flashing lights are visualised, and the user can also use the mouse to click at objects to produce input to the simulation, e.g., press a window to start a glass break detector.

- In [11] CP-nets are used to model descriptions of services in an Intelligent Network. The formal descriptions of telecommunication services are visualised using graphical feedback from simulations of the CPN model. Consistently with the MAFIA project the aim of the choice of graphics is to visualise the behaviour in a way that is easily understandable and natural to people from the problem domain (here Intelligent Networks) who are not familiar with CP-nets. The visualisation used in [11] is a simple picture of a number of telephones in a network. The CPN model is also extended to produce MSCs as output from simulations.

The use of domain specific graphics in [77] and domain specific graphics and MSCs in [11] differ from the use in the MAFIA project in several ways: To reflect the different needs of different user groups in the MAFIA project the level of detail used in the MSCs can be dynamically changed during simulations. Furthermore, the MSCs are extended to incorporate the use of domain specific graphics, thus, providing a a link between the communication between features and the appearance of the UI of the mobile phone.

Chapter 6

Conclusions and Directions for Future Work

This chapter concludes on the work done as a part of this thesis and presents directions for future work. Section 6.1 summarises the contributions of the four papers constituting part II of the thesis. Section 6.2 discusses possible directions for future work.

6.1 Summary of Contributions

The research work done as part of this thesis consider the symmetry method in the context of CP-nets. Symmetry in CP-nets is specified as permutations of the atomic colour sets. Hence, the term *permutation symmetry* is used to denote symmetries of CP-nets. Condensed state spaces are also called *state spaces with permutation symmetries* (SSPSs).

The main motivation behind the research work done as a part of this thesis has been development of theoretical aspects as well as computer tools to improve the practical applicability of the symmetry method for CP-nets. It has been a deliberate, and in my opinion important, choice to obtain a well-balanced mixture of three aspects within the work: theory, tools, and practical use. In the following the main contributions of the four papers constituting part II of the thesis are summarised:

The Symmetry Method in Practice: Analysis of a Flowmeter System. The main contribution of the first paper [65] is the applicability of the symmetry method for the analysis of an industrial system, i.e., a flowmeter system from the Danish manufacturing company Danfoss. The research work done as a part of the Danfoss project identified the need for better tool support for the symmetry method in the context of CP-nets. Also the negative impact of the orbit problem was experienced and motivated research work concerning the development of techniques to obtain efficient algorithms for generating SSPSs for CP-nets. Another contribution of the research work done in the Danfoss project was the first step towards more automatic tool support for the symmetry method. As a part of the Danfoss project a semi-automatic consistency check of symmetry specifications was designed and implemented.

Exploiting Stabilizers and Parallelism. The main contribution of the second paper [66] is the use of algebraic techniques and parallelism to alleviate the negative impact of the orbit problem in state space generation with the symmetry method. Previous work with SSPSs for CP-nets is based on an approach where the first marking reached from an equivalence class is the marking used as the representative. [66] uses another approach based on calculations of canonical (unique) representatives for the equivalence classes. A second contribution of the research work is the specification of a canonical form for markings of CP-nets.

A third contribution of the work is the development of a tool, i.e., an integration between the Design/CPN tool and the GAP tool, which fully automates the suggested techniques. One of the strengths of the paper is that the techniques are evaluated in practice, not as isolated experiments but integrated into SSPS generation of CP-nets using the developed tool. Except for the specified canonical form the techniques are not specific to CP-nets and can be applied in other formalisms where the symmetry method apply.

State Space Generation with the Symmetry Method. The main contribution of the third paper [63] is the development of the Design/CPN OPS tool, which automatically generates the predicates expressing whether two states or actions are symmetric used in SSPS generation. Together with the Design/CPN OE/OS tool this presents fully automatic generation of SSPSs for CP-nets with consistent symmetry specifications.

Another contribution of the work is the development and evaluation of techniques to improve the run-time and memory requirements of the two predicates. The developed algorithms are integrated into the Design/CPN OPS tool.

Modelling and Analysis of Feature Interactions. The main contribution of the fourth paper [67] is the application of CP-nets for the modelling of features and feature interactions in Nokia mobile phones. An important aspect of the work done in the MAFIA project is the use of domain specific graphics for visualisation of the CP-nets. The contribution of the MAFIA project, compared to other projects where domain specific graphics is used for visualising the behaviour of programs, is the use of domain specific graphics in the MSCs as well as the level of detail which can be changed dynamically during simulations of the CPN model. The use of domain specific graphics in the MAFIA project allows several different user groups to simultaneously work with the CPN model and obtain the kind of information they are interested in. A second contribution of the research work done in the MAFIA project is the development of an alternative design for the mobile phone software system allowing feature interactions to be handled in a more consistent and reliable way. Finally, a minor contribution of the work is the use of state spaces methods and model checking techniques to do initial analysis of the CPN model and evaluate the applicability of the modelling techniques for such analysis.

The above discussions summarises the four papers constituting part II of this thesis. The three first papers contributes to the improvement of three aspects of the practical applicability of the symmetry method for CP-nets:

Theory. Development of algorithms to alleviate the negative impact of the orbit problem during calculation of the SSPSs.

Tools. Development of a tool which automates the use of the symmetry method.

Practical use. Application and evaluation of the symmetry method in an industrial setting.

The fourth paper presenting work where CP-nets are applied for the modelling and analysis of feature interactions in mobile phones. The paper is somewhat different from the first three in the sense that the it does not concern the symmetry method. The work reported in [67] as a part of the research done during my PhD studies is motivated by my interest in research in industrial settings where applicability of the research results is a central issue. In connection with the MAFIA project Nokia Research Centre hosted a very inspiring research visit where central elements from my PhD work were applied and evaluated in an industrial setting with final results, i.e., applicability of the methods, and customer satisfaction as a central element. Hence, the MAFIA project contributed as a link between the three aspects: theory, tool, and practical use.

6.2 Future Work

In the previous section the main contributions of the thesis are summarised and it is concluded that the research work done as a part of this thesis contributes to the improvement of the practical applicability of the symmetry method for CP-nets. However, from the discussions of related work in Chapters 2 - 5 it can bee seen that the symmetry method and other reduction methods for alleviating the state explosion problem is an on-going research field studied in many modelling formalisms. Hence, there are still many possible directions for future work. Below a few of them are discussed.

Storage of State Spaces. From the Danfoss project summarised in Chapter 2 it can be seen that the sizes of the state spaces that can be handled by the current implementation of the Design/CPN OG tool, i.e., the state space tool in Design/CPN, is low compared to other state-of-the-art verification tools. One of the reasons is that the concept of a state in CP-nets is very complex and thus takes up more space than states in other modelling languages. However, it is fair to say that another reason is due to the current implementation of the Design/CPN OG tool. Even though attempts are made to obtain efficient storage through the use of data structures exploiting the hierarchical structure and locality of CP-nets, the current implementation is not very space efficient. Hence, an important topic for future work is to investigate other data structures for the state space storage in Design/CPN.

With the introduction of new data structures for the storage of state spaces for CP-nets another important topic for future research is to evaluate the applicability of the different data structures in connection with state space reduction methods, e.g., the symmetry method. The specified canonical form for CP-nets in the work presented in this thesis provides a step towards the use of new data structures like Binary Decision Diagrams (BDDs). The reason this is an interesting topic for further work is that there is no direct relationship between the size of the state space and the size of the BDD which represents it. Hence, it is not clear whether BDDs presents improved data structure for state space storage when also exploiting symmetry.

Canonical Forms for CP-nets. A canonicalization function for states is a function that given a state s calculates an unique representative for the equivalence class $[s]$. A canonicalization function for markings of CP-nets is defined in [66] and summarised in Chapter 3. The calculation of the specified canonical form is quite lengthy and the calculation potentially requires a large set of permutation symmetries to be applied to the marking; this is also the case when the suggested algebraic techniques are used for the reduction. Hence, a topic for future work is definition and comparison of other canonical forms for CP-nets. When using canonicalization functions the calculated representatives of the equivalence classes are unique. A possibility is to relax this requirement and as a consequence obtain less reduction. Therefore, an interesting approach for future work is to experiment with the use of normalisation functions, i.e., functions mapping a marking M into one element of a small set of symmetric markings.

Improved Data Structures for Permutation Symmetries. Efficient group representations and manipulations are important aspects of the symmetry method. The Design/CPN OPS tool uses restriction sets to represent sets of permutation symmetries. The strength of restriction sets is that arbitrary sets of permutation symmetries can be represented as a set of restriction sets. However, a major drawback is that a compact representation is only obtained for some sets. Hence, to represent an arbitrary set of permutation symmetries of size n in worst case n restriction sets are needed. In Chapter 3 it is discussed how the facilities of the GAP tool was used to obtain an efficient calculation and representation of the stabilizer groups. However, with the suggested techniques problems are experienced when the algebraic groups of symmetries used for the reduction grows. The reason is that when integrated into state space analysis the stabilizer groups are not used directly. Instead the set of cosets are calculated. Since, the set of coset representatives form an ordinary set (and not an algebraic group) the techniques used in GAP fails to obtain a compact representation of the set of coset representatives and the memory requirements becomes a serious bottleneck in practice. Since sets of coset representatives do not have a structure that in general may benefit from the use of restriction sets, restriction sets do not seem to be a promising solution for obtaining a compact representation of sets of coset representatives. Hence, an important topic for future work is the development of data structures for compact representation and efficient manipulation of general sets of permutation symmetries. In the discussion of related work in Chapter 4 it is discussed how the lazy listing approach from [63] may be a promising direction for future work in this direction.

Improved Analysis of Feature Interactions. The research work done in the MAFIA project primarily focused on analysis by means of simulations; only a first attempt to do state space analysis was done. The state space analysis was based on verification of properties from a set of state spaces of different configurations of the CPN model, i.e., the basic model plus a number of different features. A topic for future work is development of a more automatic approach for the construction and analysis of a set of state spaces without manual user invention.

Future work could also include an automatic categorisation of the feature interactions based on behavioural properties verified from the state space. In the MAFIA project the feature interactions are categorised according to some behavioural properties, e.g., feature interactions caused by conflicting UI requirements, feature interactions caused by one feature's use of another feature etc. Currently, the categorisation is done by hand.

The MAFIA project was originally motivated by the fact that many feature interactions were not systematically specified and, hence, it was very difficult to develop suitable test programs for the mobile phone software system. A possible direction for future work is to investigate whether the results from the MAFIA project can be used to automatically derive test cases covering the feature interactions present in a CPN model.

Part II

Papers

Chapter 7

Modelling and Analysis of a DANFOSS Flowmeter System using Coloured Petri Nets

The paper *Modelling and Analysis of a* DANFOSS *Flowmeter System using Coloured Petri Nets* constituting this chapter has been published as a conference paper [65].

[65] L. Lorentsen, L. M. Kristensen. Modelling and Analysis of a DANFOSS Flowmeter System using Coloured Petri Nets. In *Proceedings of the 21th International Conference on Application and Theory of Petri Nets (ICATPN'2000)*, volume 1825 of Lecture Notes in Computer Science, pages 346–366, Springer-Verlag, 2000.

The contents of this chapter is equal to the conference paper [65] except for minor typographical changes.

Modelling and Analysis of a DANFOSS Flowmeter System using Coloured Petri Nets

Louise Lorentsen[*] Lars Michael Kristensen[*]

Abstract

DANFOSS is a Danish manufacturer of refrigeration, motion, heating, and water controls. This paper describes the main results of a project on the modelling and analysis of a DANFOSS flowmeter system using Coloured Petri Nets (CP-nets or CPNs). A modern flowmeter system consists of a number of communicating processes, cooperating to make various measurements on, e.g., the flow of water through a pipe. The purpose of the project was to investigate the application of CP-nets for validation of the communication protocols used in the flowmeter system. Analysis by means of state spaces successfully identified problems in the proposed communication protocols. An alternative design was analysed using state spaces reduced by taking advantage of the inherent symmetries in the system. Exploiting the symmetries made it possible to analyse configurations of the flowmeter system approaching the size of typical flowmeter systems.

7.1 Introduction

The Danish company DANFOSS is one of the largest industrial groups in Denmark, and it is one the world leaders within the area of refrigeration, motion, heating, and water controls. This paper presents the main results of a project focusing on modelling and analysis of a DANFOSS flowmeter system. The project was carried out as a joint project between DANFOSS INSTRUMENTATION, which is a subgroup of DANFOSS, and the CPN group at the University of Aarhus.

Flowmeters are primarily used to make measurements on the flow of water through pipes. The concrete flowmeter system studied in the project consisted of several processes each doing measurements on the flow of water. Examples are processes measuring the amount of water flowing through the pipe, processes measuring the temperature of the water, and processes doing calculations based on measurements obtained by other processes.

DANFOSS has in recent years changed the architecture in the flowmeter systems from a centralised solution to a more flexible and distributed solution. The main advantage of the distributed architecture is that it allows the flowmeter systems to be adapted to the specific needs of customers. However, the distributed architecture also makes new problems arise. In the distributed solution

[*]Department of Computer Science, University of Aarhus, Aabogade 34, DK-8200 Aarhus N. DENMARK, E-mail: {`louisel,kris`}`@daimi.au.dk`.

it is much more difficult to reason about the individual processes and their influence on each other. Practical tests at DANFOSS have shown that the process communication in the first design of the flowmeter system contained at least one deadlock, and it was therefore realised that a more thorough investigation of the proposed designs was needed.

The overall aim of the project reported on in this paper was to demonstrate the use of Coloured Petri Nets (CP-nets or CPNs) [46, 58] and its supporting Design/CPN tool [26] for investigating whether the different design alternatives have the desired properties as formulated by the producer. An example of such a property is the absence of deadlocks in the flowmeter system. CP-nets have previously been used in a number of projects in an industrial setting. Examples of this include the modelling of software architectures for mobile phones at Nokia [94], validation of communication protocols used in Bang & Olufsens's Beolink Audio/Video systems [18], and communication gateways at Australian Defence Forces [31]. It was therefore envisioned that CP-nets would also be applicable for raising the quality of the design process for the new flowmeter system at DANFOSS.

The paper is organised as follows. Section 7.2 gives an overview of the project organisation and the different activities. Section 7.3 contains an introduction to the DANFOSS flowmeter system. Section 7.4 presents selected parts of the CPN model of the flowmeter system. Section 7.5 describes how state spaces were used to investigate the correctness of the flowmeter system. Section 7.6 describes a new design proposal and explains how it was verified using state spaces reduced by means of symmetries. Finally, Sect. 7.7 contains the conclusions. The reader is assumed to be familiar with the basic ideas of High-level Petri Nets and state spaces (also called occurrence graphs or reachability graphs/trees).

7.2 Overview of the Project

The project was organised in three phases and involved engineers from DANFOSS and people from the CPN group, including the authors of this paper. Hence the project group consisted of persons with expertise in the application domain, i.e., the flowmeter system, as well as members with expertise in the methods and tools to be applied, i.e., CP-nets and Design/CPN. The project was divided into three main phases which altogether ran over a period of 15 months. There have however been gaps between the different phases which means that the project ran actively for a period of 7 months.

Modelling. The first phase of the project focused on the construction of a CPN model of the flowmeter system. An initial CPN model captured two different design proposals for the flowmeter system. Overview information was provided at meetings with engineers at DANFOSS and complemented by internal documentation about the flowmeter system made available by DANFOSS. The primary purpose of the meetings at this stage was to obtain an overview of the flowmeter system and discuss the main issues related to the design. Throughout the first phase, focus gradually shifted from getting information about the flowmeter system and understanding the design issues to discussions of the CPN models which were constructed by the people from the CPN group. The main purpose of these discussions was to ensure that the CPN model correctly reflected the two different design proposals. Interactive simulations, i.e., single step simulations with detailed graphical feedback, were used to investigate the behaviour of the CPN model as well as for debugging purposes.

Using the Message Sequence Charts library [16], extensions were made to the CPN model allowing Message Sequence Charts (MSCs) [45] to be automatically constructed as graphical feedback from simulations. During project meetings these interactive simulations and MSCs were used to discuss and review the behaviour of the CPN model. Combined with the graphical nature of Petri Nets this mediated the identification of a number of discrepancies between the design and the CPN model. Subsequent to the meetings the CPN model was modified according to the identified discrepancies. Eventually these reviews lead to a validated CPN model in the sense that it correctly captured what was considered to be the relevant aspects of the two design proposals. The reason for choosing MSCs to visualise the behaviour of the CPN model was that diagrams very close to MSCs were already used in the design process at Danfoss. This allowed the behaviour of the CPN model to be visualised in a way that was very familiar to the engineers from Danfoss.

State Space Analysis. In the second phase of the project, focus changed from modelling to state space analysis [47]. The analysis was done by means of the Design/CPN Occurrence Graph Tool (OG Tool) [26]. The aim of the second phase was to investigate whether the design alternatives had the desired properties specified by Danfoss. In this phase we encountered the *state explosion problem*, i.e., the state spaces started to grow rapidly when analysing configurations of flowmeter systems with more than three processes. However, even the analysis of small configurations of the flowmeter system identified deadlocks and problems with the consistency of data in the two design proposals.

Analysis by meas of State Spaces with Permutation Symmetries. In the third phase of the project, a modified design of the flowmeter system was analysed. The CPN model was revised to capture the modified design and by means of state space analysis it was verified that the desired properties of the flowmeter system were fulfilled for small configurations of the flowmeter system. To be able to verify larger configurations, symmetries in the flowmeter system were exploited. The symmetries in the flowmeter system made it possible to apply state spaces reduced by means of permutation symmetries [47, 48] to alleviate the state explosion problem. The analysis was done by means of the Design/CPN OE/OS Graph Tool [26]. Exploiting the symmetries made it possible to analyse configurations of the flowmeter system approaching the size of typical flowmeter systems.

7.3 The Danfoss Flowmeter System

This section presents the Danfoss flowmeter system modelled in the project. The overall architecture of the system is described first, followed by a description of the communication protocols. We describe the main issues in the design of the flowmeter system, and finally we list some of the most crucial properties which the flowmeter system is required to fulfil.

7.3.1 System Architecture and Communication Protocols

Figure 7.1 shows the overall architecture of the flowmeter system. A flowmeter system consists of one or more *modules* connected via a *Controller Area Network* (CAN) [62]. Each module consists of a number of processes called *CAN Applications* (CANAPPs) and a *driver* that interfaces the module to the CAN. Figure 7.1 shows an example of a flowmeter system consisting of three modules containing two, three, and four CANAPPs, respectively. Each CANAPP in the system has a small piece of local memory which holds a number of so-called *attributes*. The communication in the system consists of asynchronous message passing between the CANAPPs. This message passing allows each CANAPP to read and write the attributes of the other CANAPPs.

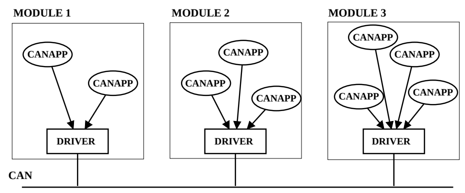


Figure 7.1: Overall architecture of the flowmeter system.

The concrete location of the CANAPPs and the number of modules are flexible, e.g., in a system with four CANAPPs it is possible to put each of the CANAPPs in an individual module or to have two modules with two CANAPPs each. A typical flowmeter system consists of 3-10 CANAPPs and 1-5 modules depending on the location of the CANAPPs. It is, however, important to mention that the location of the CANAPPs is fixed during the operation of the flowmeter system, i.e., it is not possible for one CANAPP to migrate from one module to another module. In the rest of this paper we will use the notation $\text{CANAPP}_{(i,j)}$ to denote CANAPP j on module i. Similarly, we will use Driver_j to denote the driver on module j, and Module_j to denote module j.

The communication between the CANAPPs is based on a protocol architecture with three layers. The layers constitute a collapsed form of the OSI seven layer architecture, mapping onto the physical, data link, and application layers of the OSI Reference Model [25].

All communication in the flowmeter system consists of asynchronous message passing between the CANAPPs. We will illustrate a representative communication pattern shortly. The different message types are listed in Table 7.1. Basically the messages can be divided into two groups depending on their

Table 7.1: The message types and their function.

Message	Function
ReadRequest	Request to read an attribute of another CANAPP
ReadResponse	Response to a ReadRequest containing the value of an attribute
WriteRequest	Request to write an attribute of another CANAPP
WriteResponse	Response to a WriteRequest indicating a change of an attribute
Broadcast	Distribute a value to all other CANAPPs in the system (without acknowledgement from each driver)
Event	Distribute a value to all other CANAPPs in the system (with acknowledgement from each driver)

use. The *Read* messages (ReadRequest and ReadResponse) and the *Write* messages (WriteRequest and WriteResponse) are used in point-to-point communication between the CANAPPs, whereas the Broadcast and Event messages are used in broadcast communication. In point-to-point Read (Write) communication, a ReadRequest (WriteRequest) is sent to read (write) the attribute of another CANAPP. A ReadResponse (WriteResponse) is then generated by the receiving CANAPP containing either the value of the attribute (in case of a read message) or just a value indicating an accept (in case of a write message). The delivery of the point-to-point messages is guaranteed by use of an acknowledgement mechanism in the communication between the drivers. The Broadcast and Event messages are used to distribute information to all CANAPPs in the flowmeter system – either as a Broadcast message without any guaranty of delivery, or as an Event message, with guaranteed delivery.

A typical point-to-point communication in a flowmeter system consisting of two CANAPPs located in different modules is shown in the MSC in Fig. 7.2. The MSC illustrates a write communication between two CANAPPs located in different modules. The MSC is identical to the kind of MSCs used intensively in phase one of the project.

The MSC contains a vertical line for each of the two CANAPPs, the drivers of the modules, and the CAN. The arrows between the vertical lines correspond to messages sent in the flowmeter system. The communication sequence considered corresponds to a write request/response communication to $\text{CANAPP}_{(2,1)}$ initiated by $\text{CANAPP}_{(1,1)}$, and causes the following sequence of events to occur. The numbers in the list below correspond to the numbers found below the arrows and next to the mark on the rightmost line of Fig. 7.2.

1. $\text{CANAPP}_{(1,1)}$ generates a WriteRequest message to be sent to $\text{CANAPP}_{(2,1)}$ to write the value of an attribute, and passes the message to Driver_1.

2. Driver_1 sends the message on the CAN.

3. Driver_2 receives the message from the CAN.

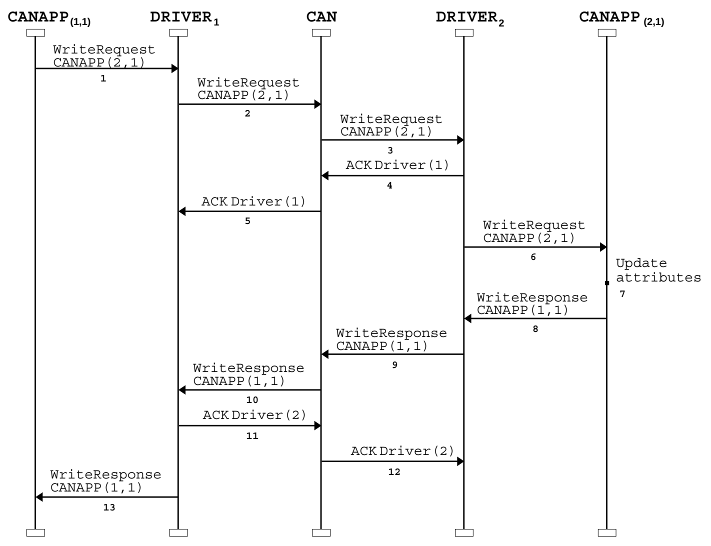

CANAPP(1,1)
DRIVER1
CAN
DRIVER2
CANAPP (2,1)
WriteRequest CANAPP(2,1) 1
WriteRequest CANAPP(2,1) 2
WriteRequest CANAPP(2,1) 3
ACK Driver(1) 4
ACK Driver(1) 5
WriteRequest CANAPP(2,1) 6
Update attributes 7
WriteResponse CANAPP(1,1) 8
WriteResponse CANAPP(1,1) 9
WriteResponse CANAPP(1,1) 10
ACK Driver(2) 11
ACK Driver(2) 12
WriteResponse CANAPP(1,1) 13

Figure 7.2: Write communication based on the request/response mechanism.

4. Driver$_2$ generates an acknowledgement which is sent on the CAN.
5. Driver$_1$ receives the acknowledgement from the CAN.
6. Driver$_2$ delivers the WriteRequest message to CANAPP$_{(2,1)}$, which accepts to have its attribute written by CANAPP$_{(1,1)}$.
7. CANAPP$_{(2,1)}$ updates the requested attribute.
8. CANAPP$_{(2,1)}$ now generates a WriteResponse message indicating that the attribute has been updated. The response is handed to Driver$_2$.
9. Driver$_2$ sends the message on the CAN.
10. Driver$_1$ receives the WriteResponse message from the CAN.
11. Driver$_1$ generates an acknowledgement which is sent on the CAN.
12. Driver$_2$ receives the acknowledgement from the CAN.
13. Driver$_1$ delivers the WriteResponse message from CANAPP$_{(2,1)}$ to CANAPP$_{(1,1)}$.

7.3.2 CANAPP Design Patterns

When two CANAPPs communicate they do it in an asynchronous way as illustrated above. When receiving a ReadRequest (WriteRequest) a ReadResponse (WriteResponse) is generated and sent back to the sender of the request. When designing the flowmeter system Danfoss used the Octopus method [5]. This method describes two different design alternatives/patterns for such asynchronous communication between objects (processes). The two design alternatives are called *Internal Wait Point* and *Primary Wait Point*. Below we briefly describe each of the two design patterns.

Internal Wait Point (IWP) approach. When a CANAPP sends a request the execution of the CANAPP is blocked. If the CANAPP is located in the same module as other CANAPPs, then all CANAPPs in the module are blocked. The CANAPPs are not released until a response matching the request has been received.

Primary Wait Point (PWP) approach. In this approach the response message is treated as an event. The requesting CANAPP and all other CANAPPs residing in the same module are not blocked as in the IWP approach. This means that a CANAPP can receive a request from another CANAPP even if it is temporarily waiting for a response to a previously sent request. This is typically implemented by means of two threads.

As identified in practical tests at Danfoss, the CANAPPs need to be carefully designed, e.g., to avoid deadlocks. This leads to the formulation of three crucial properties which the final design of the flowmeter system is required to posses. The properties are given here in an informal way. We will show in Sect. 7.5 how they can be translated into dynamic properties of the constructed CPN model.

Absence of Deadlocks. It should not be possible to bring the flowmeter system into a situation in which all the CANAPPs in the flowmeter system are blocked.

Absence of Attribute Corruption. It is important for the correct operation of the flowmeter system that when a CANAPP has initiated a request its attributes are not modified before the request has been completed.

Topology Independence. One of main advantages envisioned for the flowmeter system was that it could easily be adapted to customers' needs by providing flexibility in the choice as to which and how many CANAPPs should go into the modules. It is therefore important that the two properties above are valid independently of how the CANAPPs are distributed in the modules.

7.4 CPN Model of the Flowmeter System

This section presents selected parts of the CPN model of the flowmeter system. The purpose of this section is twofold. Firstly, to provide an overview of the CPN model, and secondly, to give an idea of the complexity of the CPN model and the abstraction level chosen. The CPN model has been put together in such a way that it captures both a design based on the Primary Wait Point (PWP) approach and a design based on the Internal Wait Point (IWP) approach. The configuration, i.e., the distribution of CANAPPs, is captured in the initial marking. Hence, the two design alternatives and different configurations of the flowmeter system can be analysed using only one CPN model but with different initial markings.

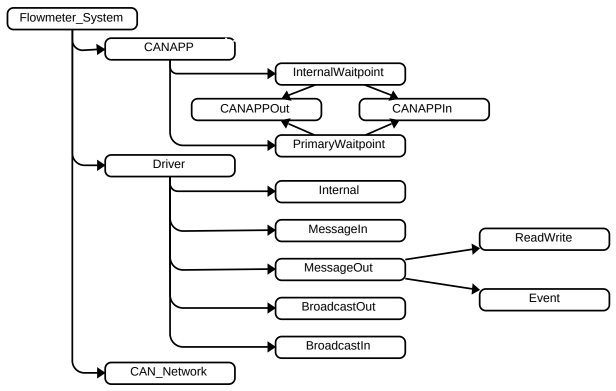


Figure 7.3: The hierarchy page.

7.4.1 CPN Model Overview

Figure 7.3 gives an overview of the CPN model by showing how it has been hierarchically structured into 15 modules (subnets). The subnets of the model are in CPN terminology referred to as *pages*. Each node in Fig. 7.3 represents a page of the CPN model. An arc between two nodes indicates that the source node contains a so-called *substitution transition* whose behaviour is described on the page represented by the destination node.
The CPN model consists of three main parts. One part modelling the CANAPPs consisting of the pages CANAPP, PrimaryWaitPoint, InternalWaitPoint, CANAPPIn, and CANAPPOut. A second part modelling the drivers consisting of the pages Driver, Internal, MessageIn, MessageOut, ReadWrite, Event, BroadcastOut, and BroadcastIn. The third part modelling the CAN consists of page CAN_Network.

Page Flowmeter_System, depicted in Fig. 7.4, is the top-most page of the CPN model and provides the most abstract view on the CPN model. The page consists of three substitution transitions corresponding to the three layers of the protocol architecture of the flowmeter system. Between each of the layers there are a number of places modelling the buffers between the layers. The detailed behaviour of CAN_Network, Driver, and CANAPP is modelled on subpages associated with the substitution transitions. In the following we will explain in more detail how the CANAPPs are modelled in the design based on the PWP approach. The modelling of the IWP approach is similar in complexity.

7.4.2 Modelling of the CANAPPs

Figure 7.5 depicts the page PrimaryWaitPoint which is the top-most page in the part of the model concerned with the CANAPPs in the PWP approach. The modelling of the CANAPPs has been split in two parts. The part of the CANAPP responsible for sending requests and receiving responses is modelled by the substitution transition CANAPPOut. The part of the CANAPP responsible for handling incoming requests and generating responses is modelled by the substitution transition CANAPPIn. The places in the lower part of the page model the buffers between the CANAPP layer and the driver layer. The two Idle places are used to model the initial state of the two threads in the CANAPP.

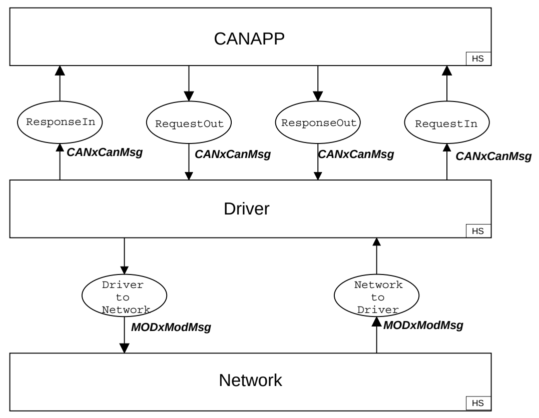


Figure 7.4: The page Flowmeter_System.

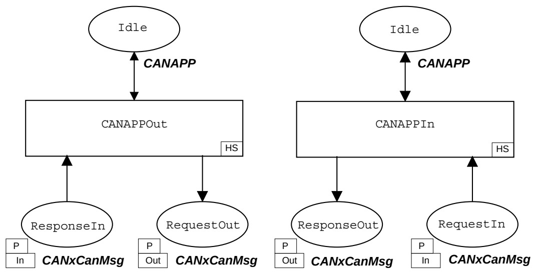


Figure 7.5: The page PrimaryWaitPoint.

Figure 7.6 depicts the page CANAPPOut which is the subpage of the substitution transition CANAPPOut shown in Fig. 7.5. This page is an example of a page at the lowest level of the CPN model. It models the control flow in the part of the CANAPP generating requests to the other CANAPPs and awaiting responses. The sending of a request is modelled by the transition Request, which causes the CANAPP to change its state from being Idle to Waiting, and pass the message to the driver by putting it into the buffer modelled by the place RequestOut. This corresponds to event 1 in Fig. 7.2. The driver in the module will remove the message from the place RequestOut and deliver it to the destination. When the response returns, the corresponding message is put in the buffer modelled by the place ResponseIn. This corresponds to events 2-12 in Fig. 7.2. The actual reception of a response is modelled by the transition Confirm. An occurrence of this transition removes the response from the place ResponseIn, updates the attributes of the CANAPP, modelled by place Attributes, and causes the CANAPP to change its state from Waiting to Idle. This corresponds to event 13 in Fig. 7.2.

The places Config and Services are used to model configuration information which can be accessed by the CANAPP. Page CANAPPIn, modelling the handling of an incoming request and the sending of a response, is similar to the CANAPPOut page. The two parts of the CANAPPs run in parallel reflecting that in the PWP approach the CANAPPs are able to make outgoing requests

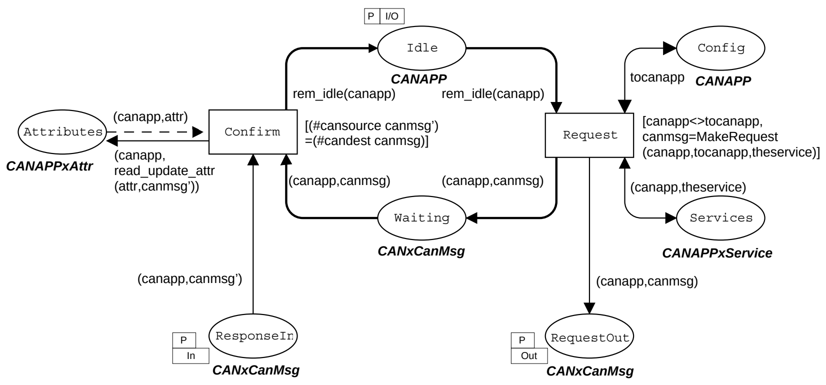


Figure 7.6: The page CANAPPOut.

as well as handle incoming requests concurrently.

7.5 Analysis of two Initial Design Proposals

This section describes how the two design proposals based on the PWP and the IWP approaches have been analysed by means of state spaces and the Design/CPN Occurrence Graph Tool (OG Tool) [26]. This section also presents the modifications made to the CPN model in order to make it suited for state space analysis, it states the goals of the analysis, and it presents the obtained results.

To make the CPN model suited for state space analysis some adjustments were needed. Essentially two modifications of the CPN model were made. The modelling of the attributes in the CANAPPs was simplified such that the concrete values of the attributes were no longer modelled. This simplification is justified since no parts of the CPN model make choices depending on concrete values of the attributes. When the concrete values of the attributes are not modelled, then the read and write messages become similar since their effect is no longer modelled. The Broadcast and Event messages were not considered. Handling these messages in the context of state space analysis would require more extensive modifications to the CPN model in order to obtain a CPN model with a finite state space. It was therefore decided to leave these two kinds of messages out.

7.5.1 Analysis Goals

The primary goal of the state space analysis was to investigate whether the two design proposals fulfilled the three requirements stated in the end of Sect. 7.3. The first step in order to investigate this was to translate them into dynamic properties of the CPN model. This makes it possible to formulate the requirements as *queries* in the OG Tool. The answers to the queries can then be automatically determined by the OG Tool when a state space has been generated. Below we show how to translate each of the three properties into queries which can be invoked in the OG Tool.

Absence of Deadlocks.

This property can be formulated as the absence of reachable *dead markings* in the CPN model. A dead marking is a marking without enabled transitions, and it is an example of a standard dynamic property of a CPN model. The OG Tool has a built-in standard query function ListDeadMarkings, which lists the reachable dead markings (if such markings exist).

Absence of Attribute Corruption.

When a CANAPP has initiated a request it is not allowed to start handling an incoming request before the request has been completed, i.e., a response has been received. This property cannot easily be formulated as a standard dynamic property of the CPN model. However, it can be conveniently formulated using temporal logic [22]. The OG Tool library ASK-CTL [26] makes it possible to make queries formulated in a state and action oriented variant of CTL [13]. That the attributes cannot be corrupted for a given $\mathrm{CANAPP}_{(i,j)}$ can be expressed as the following action-based CTL formula. An explanation is given below.

$$\begin{array}{l}\mathsf{AG}((\mathsf{Request}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle) \Rightarrow \\ \mathsf{A}((\neg(\mathsf{Indication}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle))\ \mathsf{U}\ (\mathsf{Confirm}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle)))\end{array}$$

The formula states that whenever (denoted AG) the transition Request occurs in a binding corresponding to $\mathrm{CANAPP}_{(i,j)}$, then in all futures (denoted A) the transition Indication cannot occur in a binding corresponding to $\mathrm{CANAPP}_{(i,j)}$ until (denoted U) the transition Confirm has occurred in a binding corresponding to $\mathrm{CANAPP}_{(i,j)}$. An occurrence of the transition Request in a binding corresponding to $\mathrm{CANAPP}_{(i,j)}$ has been written as $(\mathsf{Request}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle)$. The binding of Indication and Confirm is written in a similar way. An occurrence of the transition Request (see Fig. 7.6) models the start of a request, an occurrence of transition Indication models the start of handling an incoming request, and an occurrence of transition Confirm (see Fig. 7.6) models the reception of a response.

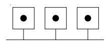
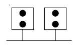
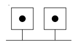
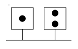
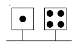
Topology Independence.

The two properties above should be valid for the system independently of how the CANAPPs are placed in the modules in the system. This property can therefore be investigated by analysing different configurations of the flowmeter system. Investigating different configurations can be done by simply changing the initial marking of the CPN model. Hence, topology independence can be investigated by constructing state spaces for different initial markings.

7.5.2 Analysis Results

State spaces have been constructed for a number of configurations of the flowmeter system. Table 7.2 gives some statistical information on the state spaces for different configurations. The Configuration column depicts the configuration in

Table 7.2: Generation statistics for full state spaces.

Configuration	WP	Nodes	Arcs	Time
	PWP	37	73	1
	IWP	6	5	1
	PWP	1,299	4,189	7
	IWP	14	13	1
	PWP	19,770	74,941	223
	IWP	11,451	37,513	109
 *	PWP	37,825	146,721	1,499
	IWP	2,266	4,841	15
	PWP	133	281	1
	IWP	80	133	1
	PWP	5,581	18,707	44
	IWP	446	869	1
	PWP	62,605	276,721	1,484
	IWP	26	25	1
 *	PWP	44,470	190,945	544
	IWP	3,866	8,473	33

question. We have used a graphical notation to indicate the configuration considered. For instance, the second row in the right-most part of Table 2 gives statistics for a configuration with two modules with one and two CANAPPs, respectively. The WP column shows which wait point approach was considered. The Nodes and Arcs columns give the number of nodes and arcs in the state space, respectively. The Time column gives the time in seconds it took to generate the state space. All state spaces were generated on a Sun Workstation with 512 MB of memory. It is worth observing that the IWP approach gives smaller state spaces than the PWP approach. The reason for this is that in the IWP approach the CANAPPs block after transmission of a message, and hence there is less concurrency between the CANAPPs compared to the PWP approach. With the available computing power and due to the state explosion problem, it was not possible to construct the full state space for large configurations in the PWP approach. Therefore, in some of the configurations only external communication (communication between CANAPPs located in different modules) is analysed. An asterisk ($*$) indicates that only external communication is analysed in the given configuration. However, even the analysis of small configurations of the flowmeter system showed that neither of the two design alternatives satisfies the required properties.

In all the configurations considered, the analysis revealed that the design based on the IWP approach had several deadlocks. Below we will concentrate on a specific deadlock situation found in the analysis of the IWP approach. The example given is in a configuration of the flowmeter system consisting of

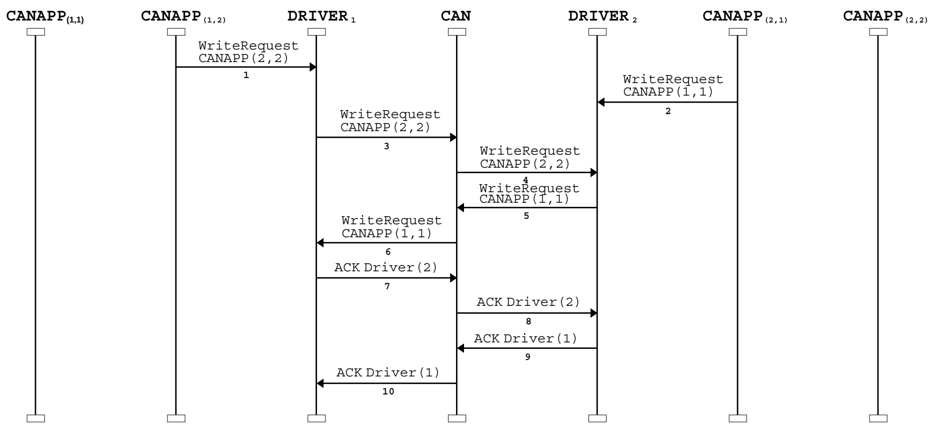

CANAPP(1,1)
CANAPP(1,2)
DRIVER1
CAN
DRIVER2
CANAPP(2,1)
CANAPP(2,2)
WriteRequest CANAPP(2,2) 1
WriteRequest CANAPP(1,1) 2
WriteRequest CANAPP(2,2) 3
WriteRequest CANAPP(2,2) 4
WriteRequest CANAPP(1,1) 5
WriteRequest CANAPP(1,1) 6
ACK Driver(2) 7
ACK Driver(2) 8
ACK Driver(1) 9
ACK Driver(1) 10

Figure 7.7: Visualisation of a path leading to a deadlock.

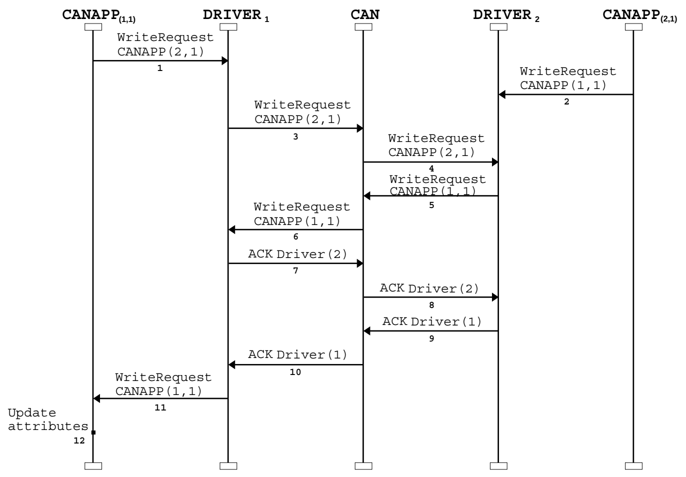
two modules each with two CANAPPs. Figure 7.7 visualises a path/execution leading to a dead marking of the CPN model. This marking was identified using the query function ListDeadMarkings followed by the use of another query function which is able to find one of the shortest paths (occurrence sequences) in the state space leading from the initial marking to a specified marking. Once such an occurrence sequence has been found, it is straightforward to visualise it using MSCs. The MSC in Fig. 7.7 shows the following sequence of events.

1. CANAPP$_{(1,2)}$ sends a WriteRequest to CANAPP$_{(2,2)}$. As the model is based on the IWP approach both CANAPP$_{(1,1)}$ and CANAPP$_{(1,2)}$ are blocked until a WriteResponse is received.

2. CANAPP$_{(2,1)}$ sends a WriteRequest to CANAPP$_{(1,1)}$. Both CANAPP$_{(2,1)}$ and CANAPP$_{(2,2)}$ are blocked until a WriteResponse is received.

3-10. The messages are sent over the CAN. All of the CANAPPs in the flowmeter system are blocked. Thus, neither CANAPP$_{(1,1)}$ nor CANAPP$_{(2,2)}$ can receive the WriteRequests, and a deadlock has occurred.

All deadlocks are of course unwanted, but especially the kind of deadlock visualised in Fig. 7.7 is problematic in a flowmeter system. The reason is that it should be possible to distribute the CANAPPs freely in the modules in the system and the concrete location of the CANAPPs should not affect the correctness of the system. If CANAPP$_{(2,2)}$ in the example above instead is placed in a separate module (if the flowmeter system consisted of three modules instead of two) no deadlock would occur for the above communication pattern. The above also illustrates topology independence as a non-trivial property of the flowmeter system. Analysis of the PWP approach showed that none of the configurations analysed had any nodes corresponding to dead markings of the CPN model. Therefore, all the configurations analysed in the PWP approach fulfil the absence of deadlocks property.



CANAPP(1,1)
DRIVER 1
CAN
DRIVER 2
CANAPP(2,1)
WriteRequest CANAPP(2,1) 1
WriteRequest CANAPP(1,1) 2
WriteRequest CANAPP(2,1) 3
WriteRequest CANAPP(2,1) 4
WriteRequest CANAPP(1,1) 5
WriteRequest CANAPP(1,1) 6
ACK Driver(2) 7
ACK Driver(2) 8
ACK Driver(1) 9
ACK Driver(1) 10
WriteRequest CANAPP(1,1) 11
Update attributes 12

Figure 7.8: Visualisation of a path leading to corruption of an attribute.

The analysis of the flowmeter system regarding the absence of attribute corruption showed that all configurations analysed for the IWP approach fulfil this property. However, the analysis in the case of the PWP approach showed that this approach does *not* fulfil the property. Figure 7.8 visualises a path/execution to a node in the state space representing a marking in which an attribute has been corrupted. The configuration of the flowmeter system considered consists of two modules each with one CANAPP. The MSC shows the following sequence of events.

1. $\text{CANAPP}_{(1,1)}$ sends a WriteRequest to write an attribute of $\text{CANAPP}_{(2,1)}$. $\text{CANAPP}_{(1,1)}$ now waits for a response but is not blocked.

2. $\text{CANAPP}_{(2,1)}$ sends a WriteRequest to write an attribute of $\text{CANAPP}_{(1,1)}$.

3-10. The messages and corresponding acknowledgements are sent over the CAN.

11. $\text{CANAPP}_{(1,1)}$ receives the WriteRequest message from $\text{CANAPP}_{(2,1)}$.

12. $\text{CANAPP}_{(1,1)}$ changes its attributes according to the value in the WriteRequest from $\text{CANAPP}_{(2,1)}$. A corruption has occurred.

To summarise the results of the analysis: the design based on the IWP approach violates the first property (absence of deadlocks) but fulfils the second (absence of attribute corruption). The design based on the PWP approach on the other hand fulfils the first property but violates the second. Thus, neither of the two design proposals are suitable for a final implementation of the flowmeter system. In the next section we will consider a variant of the PWP approach which avoids the problem of attribute corruption identified above.

7.6 Analysis of a third Design Proposal

The identification of the deadlocks and attribute corruption in the two initial design proposals naturally leads to the suggestion of a third design proposal which resolves the identified problems. In this section we present such a design proposal based on the PWP approach which overcomes the problem with attribute corruption. State space analysis of the first two design proposals had shown that the state space explosion problem prohibited analysis of larger configurations. It was therefore decided to attempt to use a state space reduction method in order to alleviate the state explosion problem when analysing the third design proposal. It was decided to use the symmetry method [47, 48] since the flowmeter system is made up of a number of identical components (the modules, drivers, and CANAPPs) whose behaviour are identical. Moreover, the symmetry method is supported by the Design/CPN OE/OS Graph Tool (OE/OS Tool) [26], and as we will see, the method preserves the properties which we want to verify. In this section we briefly present the idea in the modified design, and we then explain how it was analysed using state spaces reduced by taking advantage of the symmetries in the flowmeter system.

The basic idea in the new design proposal is to introduce a possibility for the CANAPPs to send a negative response to a request message if the CANAPP is currently waiting for a response to a previously sent request. Therefore, if the CANAPP is not in the process of performing a request when the incoming request arrives, it will access the attribute and send a positive response. If the CANAPP is in the process of performing a request, then it will not access the attribute but instead send a negative response. This can be reflected in the CPN model by adding a place modelling a shared variable between the two threads of the CANAPP (see Fig. 7.5). This variable can then be used by the thread handling the incoming requests to decide whether it is safe to access the attributes.

7.6.1 Symmetry Specification

For capturing the symmetries in the flowmeter system *state spaces with permutation symmetries* (OS-graphs in [47]) were applied. The OE/OS Tool supports state spaces with permutation symmetries based on user supplied symmetry specifications. This means that the user of the tool is required to provide the permutation symmetries, and the tool then uses these as a basis for the reduction. A symmetry specification consists of assigning *symmetry groups* of permutations to the *atomic colour sets* of the CPN model. An atomic colour set is a colour set defined without reference to other colour sets. The symmetry group determines how the colours of the atomic colour sets are allowed to be permuted, and in turn induces permutation symmetries on the markings and the binding elements of the CPN model. The idea of state spaces with permutation symmetries is to construct equivalence classes of markings which are symmetric in the sense that they can be obtained from each other by one of the permutation symmetries. Instead of representing all markings it suffices to store a representative for each equivalence class containing a reachable marking.

A similar remark applies to binding elements. In this way a condensed state space is obtained which is typically orders of magnitude smaller than the full state space.

For the flowmeter system the symmetry specification should capture the symmetry in the CANAPPs as well as in the modules. To illustrate the symmetry specification applied for the flowmeter system consider Fig. 7.9. Figure 7.9 shows the initial part of the state space for a flowmeter system consisting of two modules each containing two CANAPPs. Communication between CANAPPs in the same module has been disabled for presentation purposes. Node 1 corresponds to the initial marking and has eight immediate successor nodes corresponding to the possible requests which can be initiated in the system (each CANAPP can initiate a request to the two CANAPPs on the other module). For node n we will denote the corresponding marking M_n.

For the nodes 2, 3, 7, and 9 we have indicated in the associated dashed box what communication has been initiated, e.g., node 9 corresponds to a state of the system in which CANAPP$_{(1,1)}$ has initiated a request towards CANAPP$_{(2,1)}$. The symmetry specification is based on the observation that M_9 is symmetric to M_7 except for a permutation which swaps the two CANAPPs in Module$_2$, In a similar way it can be observed that M_2 can be obtained from M_9 by a permutation which swaps the two modules, and M_3 can be obtained from M_9 by a permutation which swaps the two modules and which swaps the two CANAPPs in Module$_2$. Furthermore, it is possible to obtain M_4, M_5, M_6, and M_8 from M_9 by permutation of the CANAPPs and the modules. If such symmetric markings are grouped into equivalence classes, it is possible to represent this initial fragment of the full state space by the condensed state space shown in Fig. 7.10. Here node 1 represents the equivalence class containing the initial marking only, and node 2 represents the equivalence class containing the markings M_2 to M_9. The marking M_9 can be chosen as a representative for this equivalence class. The successors of M_9 are grouped into equivalence classes in a similar way. Altogether this means that the 65 markings which can be reached in two steps from the initial marking can be represented using only 8 nodes in the condensed state space. In summary, the symmetry specification used for the flowmeter system allows permutation of CANAPPs within the same module, and it allows permutation of modules containing the same number of CANAPPs.

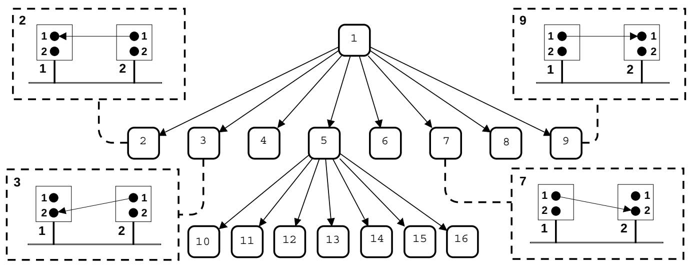


Figure 7.9: Initial fragment of the full state space.

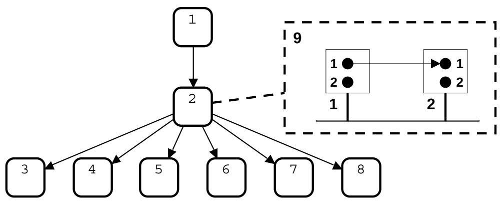


Figure 7.10: Initial fragment of the condensed state space.

7.6.2 Consistency Check

Since the OE/OS Tool supports user supplied symmetry specifications, a *consistency check* is needed to ensure that the symmetries supplied are symmetries which are actually present in the CPN model. This amounts to checking that the initial marking, the guards, and the arc expressions of the CPN model are symmetric in a way precisely defined in [47]. The currently released version of the OE/OS Tool does not support an automatic check for consistency, but as part of the project we have developed an extension of the tool which supports a semi-automatic check for consistency. It is based on a combination of syntactical and semantical checks. The semantic check is based on evaluating net inscriptions in all possible bindings. As a consequence the semantic check is time-consuming, but together with the syntactical check it is possible to make a fully automatic check for consistency for the CPN model of the flowmeter system.

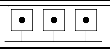
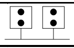
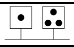
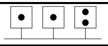
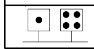
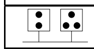
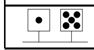
7.6.3 Analysis Results

The analysis focuses on the same three properties as for the two initial design proposals. For investigating the absence of dead markings a built-in query function of the OE/OS Tool was used which lists the equivalence classes containing the reachable dead markings (if such equivalence classes exist). The query related to the absence of attribute corruption has to be modified to take into account that the transition Indication corresponding to the reception of an incoming request can now occur in two modes depending on whether the request is accepted or rejected. The modified query is shown below and is identical to the original query except that it is only required that the Indication transition does not occur in a binding where the request is accepted, i.e., the variable mode is bound to accept.

$$\begin{aligned}&\mathsf{AG}((\mathsf{Request}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle) \Rightarrow\\ &\quad \mathsf{A}((\neg(\mathsf{Indication}, \langle canapp = \mathrm{CANAPP}_{(i,j)}, mode = accept\rangle))\ \mathsf{U}\\ &\qquad (\mathsf{Confirm}, \langle canapp = \mathrm{CANAPP}_{(i,j)}\rangle)))\end{aligned}$$

Another problem which has to be resolved with the above query is that it refers to occurrence sequences related to a specific CANAPP. Since we allow permutations of CANAPPs this property is not a priori preserved by the symmetry reduction. However, it was shown in [23, 28] that symmetry reduction preserves the truth value of a CTL formula if the truth value of its atomic state

Table 7.3: Generation statistics for full and condensed state spaces.

Configuration	Full State Space				Condensed State Space				Ratio	Sym
	Nodes	Arcs	Time	T/N	Nodes	Arcs	Time	T/N		
	19,970	74,941	223	0.011	3,410	12,478	121	0.035	5.7	6
	62,605	276,721	1,484	0.024	2,776	4,767	327	0.118	22.5	24
	295,965	1,329,113	–	–	37,156	166,316	4,123	0.111	7.9	8
	319,337	1,460,785	–	–	54,019	245,139	2,523	0.047	5.9	6
 *	456,174	2,148,585	–	–	114,370	537,857	7,428	0.065	3.9	4
 *	44,470	190,995	544	0.012	2,971	11,075	197	0.066	14.9	24
 *	578,376	2,746,401	–	–	49,848	234,431	15,214	0.305	11.6	12
 *	209,629	1,044,361	–	–	4,759	18,699	1,438	0.302	44.0	120

propositions are invariant under the permutation symmetries. The absence of attribute corruption can therefore be checked from the symmetry reduced state space by checking the property for a CANAPP$_{(i,j)}$ for which permutation is not allowed. CANAPPs which cannot be permuted are present in configurations of the flowmeter system containing a module with a single CANAPP. An alternative approach to this is to strengthen the symmetry specification such that the CANAPP in question cannot be permuted.

Table 7.3 gives some statistical information on the generation of the condensed state space for some selected representative configurations. For comparison it also gives the size of the full state space. The size of the full state space has been calculated from the condensed state space in the cases where the full state space could not be generated given the available computing power. The Time column gives the time in seconds it took to generate the state space. T/N gives the processing time (in seconds) per node in the state space. The Ratio gives the reduction obtained in the number of nodes. The column Sym gives the number of permutation symmetries for the configurations, which is an upper limit on the reduction which can be obtained. The condensed state spaces were generated on a Sun Workstation with 512 MB of memory.

As can be seen from Table 7.3, the use of symmetries yielded significant reductions in the size of the state spaces and allowed us to consider configurations of the flowmeter system which could not be handled with full state spaces. This is what we would have expected. It is, however, also worth observing that the generation of condensed state spaces was faster than the generation of the full state spaces. Even though we only have three observations, they indicate what seems to be a general fact: the time that is lost on a more expensive test on equivalence of markings and binding elements, is accounted for by having fewer nodes and arcs to generate; and also to compare with before a new node or arc can be inserted in the state space.

We were able to verify the absence of reachable dead markings in all configurations for which a condensed state space could be generated. Moreover, the absence of attribute corruption was verified for the configurations with a module containing a single CANAPP. In addition to these properties we were able to verify that the initial marking was a *home marking*. This means that from any reachable marking it is always possible to return to the initial marking. This is a very attractive property since it means that the flowmeter system can never be brought in a situation in which the system cannot be made to reenter its initial state. Moreover, it could be verified that the binding elements corresponding to the initiation of a request, reception of a response, handling of a request, and transmission of a response were *live*. This means that all CANAPPs in the system always have the possibility of completing requests and handling incoming requests.

7.7 Conclusions

In this paper we have presented the main results of a project in which CP-nets was put into practical use in an industrial setting at Danfoss for the modelling and analysis of a flowmeter system. The project was divided into three phases. During the first phase the graphical nature of Petri Nets and the capability to visualise the behaviour of a CPN model were extremely important tools in the process of validating that the CPN model correctly reflected the intended design of the flowmeter system. This observation is consistent with observations made in other industrial projects such as [18].

The second phase clearly showed the practical limits of using full state spaces in an industrial setting, since the size of the state space started to grow rapidly once larger configurations were considered. Given the available computing power it was only possible to analyse configurations with up to 3-4 CANAPPs. However, even the analysis of small configurations identified errors in the proposed designs. This also demonstrates that errors in systems tend to manifest themselves in even small configurations of the system.

The application of the symmetry method made it possible to verify configurations of the flowmeter system containing up to six CANAPPs. Since concrete configurations may contain more CANAPPs (typically up to 10), it is relevant to ask whether anything could have been done to handle even larger configurations, i.e., whether other reduction methods could have been applied in addition to the symmetry method. Since the CANAPPs in the system are asynchronous it would be obvious to consider the stubborn set method [88]. It was proved in [89] that the stubborn set method can be combined with the symmetry method. The stubborn set method was however not applied in the project since the tool support for stubborn sets in Design/CPN is currently not mature enough to be used on models of the size considered in this project.

In conclusion we believe that this project has demonstrated that CP-nets and the state space method may indeed be relevant and valuable methods to be used in the design process of not only the flowmeter system but also in future products at Danfoss which was the main goal of the project.

Acknowledgements. We would like to thank Danfoss Instrumentation, especially Arne Peters, for their contributions to this project.

Chapter 8

Exploiting Stabilizers and Parallelism in State Space Generation with the Symmetry Method

The paper *Exploiting Stabilizers and Paralellism in State Space Generation with the Symmetry Method* constituting this chapter has been published as a conference paper [66].

[66] L. Lorentsen, L. M. Kristensen. Exploiting Stabilizers and Paralellism in State Space Generation with the Symmetry Method. In *Proceedings of the Second International Conference on Application of Concurrency to System Design (ICACSD'2001)*, pages 211–220,IEEE 2001. 2000.

The contents of this chapter is equal to the conference paper [66] except for minor typographical changes.

Exploiting Stabilizers and Parallelism in State Space Generation with the Symmetry Method

Louise Lorentsen[*] Lars Michael Kristensen[†]

Abstract

The symmetry method is a main reduction paradigm for alleviating the state explosion problem. For large symmetry groups deciding whether two states are symmetric becomes time expensive due to the apparent high time complexity of the orbit problem. The contribution of this paper is to alleviate the negative impact of the orbit problem by the specification of canonical representatives for equivalence classes of states in Coloured Petri Nets, and by giving algorithms exploiting stabilizers and parallelism for computing the condensed state space.

8.1 Introduction

State space exploration has proven powerful for investigating the correctness of concurrent systems. Unfortunately, state spaces of systems tend to grow rapidly when systems become bigger. This well-known phenomenon known as the *state explosion problem*, represents a serious limitation to the use of state space methods for analysis of real-life systems.

Many techniques and methods for alleviating the state explosion problem have been suggested, such as the *symmetry method* [23, 28, 48]. The basic idea behind the symmetry method is to exploit that many concurrent systems exhibit symmetry. For example, many concurrent systems are composed of similar components whose identities are interchangeable from a verification point of view. This kind of structural symmetry is also present in the state space of such systems. The basic idea behind the symmetry method is to factor out this symmetry by grouping symmetric states into equivalence classes, and thereby obtain a *condensed state space* which is typically orders of magnitude smaller than the ordinary full state space, but from which the same behavioral properties can be verified without unfolding the condensed state space to the full state space.

Several variants of the symmetry method have been suggested for reasoning about different classes of properties. Examples of this are a method [48] for standard dynamic properties of Coloured Petri Nets (CP-nets or CPNs) [46,58],

[*]Department of Computer Science, University of Aarhus, Denmark. E-mail: louisel@daimi.au.dk.

[†]School of Electrical and Information Engineering, University of South Australia, AUSTRALIA. E-mail: lars.kristensen@unisa.edu.au.

a method for safety properties [44], and methods for temporal logic properties expressed in CTL* [24,28] and LTL [30,38]. Two main issues are common to all of these variants of the symmetry method. One is how to determine the algebraic group of allowable permutation symmetries. Another is deciding during the generation of the condensed state space whether two states s_1 and s_2 are symmetric (equivalent), i.e., whether there is one of the allowable permutation symmetries which maps s_1 into s_2. In this paper we focus on the problem of deciding whether two states are symmetric. This is also referred to as the *orbit problem*.

The computational complexity of the orbit problem has been investigated in [24] showing that it is at least as hard as the *graph isomorphism problem* for which no polynomial time algorithm is known. The results in [24] were later extended in [21] showing that the orbit problem is equivalent to important problems in computational group theory which are harder than the graph isomorphism problem. A derivate of the orbit problem is the *constructive orbit problem* which is concerned with computing a canonical (unique) representative for each equivalence class of states. The constructive orbit problem is at least as hard as the orbit problem. The computational complexity of the orbit problem and the constructive orbit problem in the context of Place/Transition nets is further investigated in [55]. It is our experience from practical experiments, e.g., in [53, 65] that the above identified time complexity is also of practical relevance as the run-time penalty incurred by the use of symmetries becomes significant as system parameters grows and the symmetry groups become large. It is therefore of importance to develop heuristics which can be used to speed-up the computation of condensed state spaces.

We consider the symmetry method in the context of CP-nets and address the orbit problem from three different angles. Firstly, we specify a canonical form for states of symmetrical CP-nets. Having such a canonical form is attractive, since it can be used to significantly speed-up the computation of condensed state spaces when the storage of the state space is based on an explicit enumeration as is the case with many state space tools. Computing a canonical representative for states of CP-nets is non-trivial because the state information is highly structured meaning that there are complicated dependencies between the entries in the state vector. This makes it difficult to identify cases of practical interest where the constructive orbit problem is easy. Secondly, we show how the computation of these canonical representatives can be done by exploiting stabilizers and cosets known from algebraic and computational group theory. Thirdly, we suggest a parallel algorithm for computing the condensed state space. To evaluate the practical applicability of these algorithms we have made an implementation based on an integration between the CP-net tool DESIGN/CPN [26] and the GAP tool [34]. GAP is a general programming environment implementing a number of efficient algorithms for manipulation of algebraic groups.

We present our results in the context of CP-nets. However, it is only the specification of the permutation symmetries and the computation of canonical representatives which are specific to CP-nets. Our results on the use of algebraic group theory and parallel construction of condensed state spaces are valid also for other modeling formalisms where the symmetry method is applicable.

The rest of this paper is organized as follows. Section 8.2 recalls the basic facts of CP-nets and the symmetry method. Section 8.3 presents the canonical form for states of CP-nets. Section 8.4 gives the algorithms for the use of algebraic techniques when computing the condensed state space. Section 8.5 gives the algorithm for parallelizing the construction of the condensed state space. Section 8.6 gives some numerical data on the performance of the algorithms on some representative case studies. Finally, in Sect. 8.7 we sum up the conclusions and give a further discussion of related work.

8.2 Background

This section summarizes the basic facts of CP-nets and the symmetry method. The definitions and notation for CP-nets are given in Sect. 8.2.1 and follow closely [46,47]. Section 8.2.2 introduces the necessary background on the symmetry method. The reader is assumed to be familiar with the dynamic behavior of Petri Nets [72].

8.2.1 Coloured Petri Nets

A *multi-set* ms over a domain X is a function from X into the set of natural numbers $\mathbb{N}$. A multi-set ms can be written as a formal sum like $\sum_{x \in X} ms(x)'x$, where $ms(x)$ is the number of occurrences of the element x in ms. $|ms|$ denotes the size of the multi-set ms, i.e., the total number of elements with their multiplicities taken into account. The empty multi-set is denoted $\emptyset$. S_{MS} denotes the set of multi-sets over a domain S.

A *Coloured Petri Net* (CP-net) [46] is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where Σ is a set of *colour sets* (data types), P is a set of *places*, T is a set of *transitions*, and A is a set of *arcs*. N is a *node function* designating for each arc a *source* and *destination*. C is a *colour function* mapping each place p to a colour set $C(p) \in \Sigma$ specifying the type (colours) of *tokens* which can reside on the place p. G is a *guard function* mapping each transition t to a boolean expression $G(t)$. E is an *arc expression function* mapping each arc a into an arc expression $E(a)$. I is an *initialization function* mapping each place p into a multi-set $I(p)$ of type $C(p)_{MS}$ specifying the initial marking of the place p.

The colour sets (data types) of a CP-net can be divided into *atomic* and *structured colour sets*. An atomic colour set is a colour set defined without reference to other colour sets. Typical examples of atomic colour sets are integers, booleans, and enumeration types. A structured colour set is a colour set defined using one of the available *type constructors*. Typical type constructors in CP-nets are product for construction of tuple types, record for construction of record types, union for construction of union types, and list for construction of list types. For a CP-net CPN, we denote by $\Sigma_A \subseteq \Sigma$ the set of atomic colour sets of CPN. For a structured colour set $S \in \Sigma$, the *base colour sets* of S are the colour sets from which S is constructed.

A *marking* (state) of a CP-net is a distribution of *tokens* carrying data values (colours) on the places of the CP-net. A marking M is a mapping which for each place p yields the multi-set of tokens on p in the marking M. The multi-set of tokens present on a place p in a marking M is denoted $M(p)$. M_0 denotes the *initial marking*. Hence, for a CP-net CPN with places $P = \{p_1, p_2, \ldots, p_n\}$ the *state vector* corresponding to a marking M of CPN can be written as $(M(p_1), M(p_2), \ldots, M(p_n))$ with $M(p_i) \in C(p_i)_{MS}$ for $1 \leq i \leq n$. The set of all markings is denoted $\mathbb{M}$.

A *binding element* (t, b) is a pair consisting of a transition t and a *binding* b of data values to the *variables* of t. The set of all binding elements is denoted BE. If a binding element (t, b) is *enabled* in a marking M_1 (denoted $M_1[(t, b)\rangle$), then (t, b) may *occur* in M_1 yielding some marking M_2. This is written $M_1[(t, b)\rangle M_2$. A *reachable marking* is a marking which can be obtained (reached) by a sequence of occurrences of transitions starting from the initial marking. $[M_0\rangle$ denotes the set of reachable markings.

8.2.2 The Symmetry Method

Symmetry in CP-nets is specified by means of a *permutation symmetry specification* which assigns an algebraic *symmetry group* of permutations to each atomic colour set. A symmetry group determines how the colours of an atomic colour set are allowed to be permuted. For an example, a symmetry group may specify that all colours can be permuted arbitrarily, or that they must all be fixed, i.e., cannot be permuted. Many intermediate forms exists, e.g., all rotations of a finite, ordered atomic colour set.

A permutation symmetry specification SG determines a group Φ_{SG} of *permutation symmetries*. A permutation symmetry for SG is a mapping ϕ which assigns to each atomic colour set $S \in \Sigma_A$ a permutation $\phi_S \in SG(S)$. A permutation symmetry ϕ determines a permutation on structured colour sets by permutation of the atomic values of the structured value according to ϕ. For an example, consider a structured colour set C which is a product of two atomic colour sets A and B. The permutation ϕ_C of a pair $(a, b) \in C$ is defined as $\phi_C(a, b) = (\phi_A(a), \phi_B(b))$. This in turn induces a permutation of multi-sets, i.e., the marking of places, by defining for a multi-set $ms = \sum_{s \in S} ms(s)'s$ over a colour set S, $\phi_{S_{MS}}(ms) = \sum_{s \in S} ms(s)'\phi_S(s)$. This in turn induces a permutation $\phi_{\mathbb{M}}$ of markings: for a marking M written as a state vector $(M(p_1), \ldots, M(p_n))$ we define $\phi_{\mathbb{M}}(M)$ as the marking with state vector $(\phi_{C(p_1)_{MS}}(M(p_1)), \ldots, \phi_{C(p_n)_{MS}}(M(p_n)))$, i.e., the marking obtaining by permuting the markings (multi-sets) of the individual places. In this paper we will omit the subscript on ϕ and write the permuted marking $\phi_{\mathbb{M}}(M)$ as $\phi(M)$, and its state vector as $(\phi(M(p_1)), \ldots, \phi(M(p_n)))$. The subscript can always be determined from the colour set of the place p in question. Permutation symmetries determines permutations on binding elements in a similar way.

The construction above induces two equivalence relations – one on the set of markings ($\approx_{\mathbb{M}}$) and one on the set of binding elements ($\approx_{BE}$). Two markings M_1 and M_2 are considered symmetric (equivalent) iff there exists a permutation symmetry $\phi \in \Phi_{SG}$ such that $M_1 = \phi(M_2)$, and similarly for binding elements. That the symmetry groups are algebraic groups ensures that the relations on markings and binding elements are indeed equivalence relations. The set of all equivalence classes for $\approx_{\mathbb{M}}$ is denoted $\mathbb{M}_{\approx}$. Similarly with $\approx_{BE}$ and $BE_{\approx}$. The equivalence class of a marking/binding element x is denoted $[x]$.

Definition 8.1
The ***condensed state space*** *of a CP-net is the directed graph* (V, E)*, where:*

$V = \{\, C \in \mathbb{M}_{\approx} \mid C \,\cap\, [M_0\rangle \neq \emptyset \,\}$ *is the set of nodes.*

$$\begin{aligned} E = \{\, & (C_1, B, C_2) \in V \times BE_{\approx} \times V \mid \\ & \exists\, (M_1, (t,b), M_2) \in C_1 \times B \times C_2 \,:\, M_1[(t,b)\rangle M_2 \,\} \end{aligned}$$
is the set of edges.

□

The condensed state space has a node for each equivalence class containing a reachable marking. The condensed state space has an edge between two nodes iff there is a marking in the equivalence class of the source node in which a transition is enabled, and whose occurrence leads to a marking in the equivalence class of the destination node.

An essential aspect of calculating condensed state spaces is when reaching a new marking/binding element x during construction to check whether a marking/binding element from $[x]$ is already contained in the condensed state space.

To be able to implement the check efficiently for markings of CP-nets, we will compute a canonical representative, i.e., a unique representative for each equivalence class by invoking some function CANONICAL which given a marking M calculates the canonical representative of $[M]$. The check then amounts to transforming the new marking reached into this unique representative and then check (using ordinary equality) whether the resulting state has already been included in the condensed state space. Efficient generation of condensed state spaces is therefore highly dependent on the complexity and the number of calls of the CANONICAL function.

As for markings, we can compute a canonical representative for the equivalence class $[(t, b)]$ of a binding element (t, b) such that $M_1[(t,b)\rangle M_2$ and check using ordinary equality whether that corresponding arc already exists between the equivalence classes of M_1 and M_2. The problem of computing canonical representatives for equivalence classes of binding elements can be reduced to the problem of computing canonical representatives of markings by viewing the variables of the transition as places and the value assigned to a variable as a singleton multi-set of tokens. Given a transition t with variables $v_1, v_2, \ldots, v_n$, a binding element of (t, b) can be viewed as a vector of singleton multi-sets $(1`b(v_1), 1`b(v_2), \ldots, 1`b(v_n))$ where $b(v)$ denotes the value assigned to v in the binding b. Since transitions cannot be permuted by permutation symmetries in CP-nets, finding a canonical representative for $[(t, b)]$ is the same as finding a canonical representative for b which by the above reduction is the same as finding a canonical representative of a marking. Hence, in this paper we concentrate on computing canonical representatives for markings.

8.3 Canonical Representatives

The basic idea behind the specification of canonical representatives for equivalence classes of states in CP-nets is to define a total ordering on the markings of the CP-net. This total ordering will be determined by the total ordering on the atomic colour sets. We therefore assume that each atomic colour set $A \in \Sigma_A$ has an associated total ordering denoted $<_A$. For the atomic colour sets appearing in practice such as integers, booleans, and enumeration types defining this order is straightforward. For structured types defined using type constructors such as product, union, record, and list, a total order can be inductively defined based on the ordering of their base colour sets using, e.g., lexicographical ordering. In the rest of this section we assume that a CP-net with places $P = \{p_1, p_2, \ldots, p_n\}$ is given and that each colour set $S \in \Sigma$ has an associated total order denoted $<_S$.

First we define a total ordering on the multi-sets. We cannot use the usual ordering ($\leq$) on multi-sets over a set S defined as $ms_1 \leq ms_2 \Leftrightarrow \forall s \in S : ms_1(s) \leq ms_2(s)$ [46] since this is only a partial order. To obtain a total order on multi-sets, we define the operation NORMALISE on a multi-set. This operation sorts the elements in the multi-set $ms \in S_{MS}$ in, e.g., increasing order according to the total order $<_S$ on S and inserts them into a list of pairs over $\mathbb{N} \times S$ such that (k, s) appears in the list iff $0 < ms(s) = k$. Formally, NORMALISE(ms) is the list $[(ms(s_1), s_1)), (ms(s_2), s_2), \ldots, (ms(s_m), s_m)]$ satisfying $s_i \leq_S s_{i+1}$ for $1 \leq i < m$. Since $<_S$ is a total order on S and $<$ is a total order on $\mathbb{N}$, we can define a total order $<_{\mathbb{N}\times S}$ on the elements in the normalised lists in a similar way as for colour sets constructed using the product type constructor. Based on this we can define a total ordering on the multi-sets based on lexiographical ordering.

Definition 8.2
Let ms_1 and ms_2 be two multi-sets over a colour set S with a total ordering $<_S$. Let $[ns_{11}, ns_{12}, \ldots, ns_{1m}] =$ NORMALISE(ms_1) and $[ns_{21}, ns_{22}, \ldots, ns_{1k}] =$ NORMALISE(ms_2). The total order $<_{S_{MS}}$ on S_{MS} is defined as:

$$\begin{array}{l} ms_1 <_{S_{MS}} ms_2 \Leftrightarrow \\ \quad ((m < k) \wedge (\forall i \leq m : ns_{1i} = ns_{2i})) \vee \\ \quad (\exists i \leq m : ns_{1i} <_{\mathbb{N}\times S} ns_{2i} \wedge \forall j < i : ns_{1j} = ns_{2j})) \end{array}$$

□

Given a set of permutation symmetries $\Phi' \subseteq \Phi_{SG}$ and a multi-set ms over S, we denote by $\psi(\Phi', ms)$ the smallest element (according to $<_S$) in the set $\{ \phi(ms) \mid \phi \in \Phi' \}$, and by $\bar{\psi}(\Phi', ms) = \{ \phi \in \Phi' \mid \phi(ms) = \psi(\Phi', ms) \}$ the subset of Φ' that maps ms into the smallest element of $\{ \phi(ms) \mid \phi \in \Phi' \}$.

Based on the total ordering on multi-sets, we can now define a total order on state vectors in a similar way.

Definition 8.3
Let $M_1 = (M_1(p_1), \ldots, M_1(p_n))$ and $M_2 = (M_2(p_1), \ldots, M_2(p_n))$ be state vectors. The ordering $<_{\mathbb{M}}$ on markings is defined as:

$$M_1 <_{\mathbb{M}} M_2 \Leftrightarrow \exists i \leq n : M_1(p_i) <_{C(p_i)_{MS}} M_2(p_i) \wedge \\ \forall j < i : M_1(p_j) = M_2(p_j)$$

□

Since $<_{\mathbb{M}}$ is a total order and the equivalence classes of markings are finite, it follows that each equivalence class of markings will have a unique minimal and maximal element which can be used as the canonical representative. In the following we will denote the smallest element in the equivalence class $[M]$ of a marking M by $[M]_{min}$.

The brute-force approach for calculation of the canonical representative of a marking is simply to apply each permutation symmetry given by the permutation symmetry specification in turn and return the smallest resulting marking. Such an algorithm is shown in Fig. 8.1. Given a marking M and the group of permutation symmetries Φ_{SG} allowed by a permutation symmetry specification SG the algorithm uses the mappings ψ and $\bar{\psi}$ to compute the canonical representative $[M]_{min}$ for $[M]$. The main problem with this algorithm is that each time a marking is reached during generation of the condensed state space $|\Phi_{SG}|$ permutation symmetries are applied to the marking. Practical experiments [65], have shown that the growth of $|\Phi_{SG}|$ as a function of the system parameters becomes a serious bottleneck in the analysis of these systems. If all colours in each of the atomic colour sets can be permuted arbitrarily there are $\Pi_{S \in \Sigma_A}(|S|!)$ permutation symmetries.

```
1: **Input** M, $\Phi_{SG}$
2: $\Phi' \leftarrow \Phi_{SG}$
3: **for all** $i \in \{1, 2, \ldots, n\}$ **do**
4:     $M'(p_i) \leftarrow \psi(\Phi', M(p_i))$
5:     $\Phi' \leftarrow \bar{\psi}(\Phi', M(p_i))$
6: **end for**
7: **return** $(M'(p_1), M'(p_2), \ldots, M'(p_n))$
```

Figure 8.1: Calculation of the canonical representative.

8.4 Exploiting Stabilizers

We now show how *stabilizers* can be used to work around the orbit problem and avoid iterating through all the permutation symmetries when computing canonical representatives. When canonicalizing a marking M, the idea is to determine the set of stabilizers (self-symmetries) of M (denoted Φ_{SG}^{M}) which are the permutation symmetries mapping M to itself. Formally, $\Phi_{SG}^{M} = \{ \phi \in \Phi_{SG} \mid \phi(M) = M \}$. In Sect. 8.4.1 we show how stabilizers can be used to reduce the number of permutation symmetries that needs to be considered when computing canonical representatives and show how the *backtrack method* can be used to compute the stabilizers in a given marking. The use of the backtrack algorithm to calculate the stabilizers of a marking is originally suggested in [51]. In Sect. 8.4.2 we show how stabilizers can be used to reduce the number of markings that needs to be canonicalized.

8.4.1 Fewer Iterations

The basic observation behind reducing the number of permutation symmetries that needs to be considered is that for a marking M some of the permutation symmetries in Φ_{SG} are known to have similar effects on M, i.e., when we apply them to M we obtain the same marking. Hence, we only have to apply one of these permutation symmetries to M when computing the canonical representative [47].

It is easy to see that the stabilizers of M is a subgroup of Φ_{SG}. A *left coset* (in the rest of the paper just referred to as a coset) of Φ_{SG}^M is a set on the form $\phi \circ \Phi_{SG}^M = \{ \phi \circ \phi' \mid \phi' \in \Phi_{SG}^M \}$, where $\phi \in \Phi_{SG}$. The set of cosets form a disjoint partitioning of Φ_{SG}^M [1]. From this it follows that two permutation symmetries, ϕ' and ϕ'', from the same coset of Φ_{SG}^M maps M to the same marking, i.e., $\phi'(M) = \phi''(M)$. Hence it suffices to consider only one permutation symmetry from each of the cosets of Φ_{SG}^M when calculating the canonical representative for M. It follows from La Grange's theorem [1] that the number of coset representatives is $|\Phi_{SG}|/|\Phi_{SG}^M|$. Hence, we now only have to consider $|\Phi_{SG}|/|\Phi_{SG}^M|$ permutation symmetries when calculating the canonical representative for M.

The backtrack method [7] can be used to compute the stabilizers of a marking M [51] since the stabilizers constitutes a subgroup of Φ_{SG}. The backtrack algorithm originates from computational group theory. The backtrack algorithm searches a subgroup Ψ from the group of all permutations of the set {1,...,m} satisfying a certain property $\Pi \in [\Psi \to \{\mathsf{true}, \mathsf{false}\}]$. A prerequisite for using the backtrack algorithm is that the set $\bar{\Pi} = \{ \phi \in \Psi \mid \Pi(\phi) \}$ constitutes a subgroup of Ψ. The property that *the permutation symmetry ϕ is a stabilizer of a set S* constitutes a subgroup of the group of all permutations. Hence, we can apply the backtrack algorithm for the calculation of the stabilizers of sets. During calculation, the backtrack algorithm maintains a subgroup of Ψ and exploits that it is only necessary to test one permutation symmetry from each coset of the subgroup currently found. In the following we sketch how the backtrack algorithm can be used for the calculation of stabilizers for a marking of a CP-net [51]. For a detailed description of the backtrack algorithm we refer to [7].

Using the backtrack method, we consider the marking of each of the places of the CP-net in turn. The marking $M(p)$ of a place p is divided into disjoint sets – one set for each coefficient $c > 0$ in the multi-set $M(p)$. This set contains the colours appearing with the coefficient c in the multi-set $M(p)$. In [4] it is shown that the set of stabilizers for a marking is the intersection of the stabilizers of these sets. Hence, we can apply the backtrack algorithm for the calculation of the stabilizers for each of these sets and intersect the partial results to obtain the stabilizers of the marking [51]. During calculation the intersection is done implicitly, i.e., by reducing the initial search domain for the subsequent applications of the backtrack algorithm. Permutation symmetries that are not stabilizers for one of the sets treated cannot be stabilizers for the marking as a whole. Hence, there is no need to include such permutation symmetries in the calculation for the next set [51]. The algorithms for using the backtrack method for calculation of stabilizers of a marking for a CP-net is shown in Fig. 8.2. The algorithm refers to a number of functions.

Coefficients(ms) calculates the coefficients appearing in a multi-set ms, i.e., for $ms \in S_{MS}$ Coefficients(ms) $= \{ms(s) \mid s \in S,\ m(s) > 0\}$

CoefficientSet(c,ms) calculates the set of elements in a multi-set ms appearing with coefficient c, i.e., for $ms \in S_{MS}$ CoefficientSet(c,ms) $= \{s \in S \mid ms(s) = c\}$

StabilizerProperty(S) returns a property $\Pi \in [\Psi \rightarrow \{\mathsf{true}, \mathsf{false}\}]$ for a set S. The calculated property Π takes a permutation symmetry as argument and returns true if the permutation symmetry is a stabilizer of the set S, otherwise Π returns false.

Backtrack(Ψ,Π) is the backtrack algorithm. It takes two arguments: a subgroup Ψ of the group of all permutations and a property Π and returns the elements in Ψ satisfying Π.

When canonicalizing a marking M we only have to consider one permutation symmetry from each of the cosets of Φ_{SG}^{M}. Hence, line 2 of the algorithm in Fig. 8.1 can be changed from $\Phi' \leftarrow \Phi_{SG}$ to $\Phi' \leftarrow$ CosetReps(Φ_{SG}^{M}), where CosetReps is a function that given a subgroup of Φ_{SG} returns a representative from each of the cosets, and Φ_{SG}^{M} is computed using the algorithm given in Fig. 8.2. A technique for calculation of coset representatives of subgroups of permutation symmetries (Φ_{SG}^{M}) is given in [2], and will not be explained here.

```
1: **Input** M, $\Phi_{SG}$
2: $\Psi \leftarrow \Phi_{SG}$
3: **for all** $i \in \{1, 2, \ldots, n\}$ **do**
4:   **for all** c in Coefficients($M(p_i)$) **do**
5:     S ← CoefficientSet(c,$M(p_i)$)
6:     $\Pi$ ← StabilizerProperty(S)
7:     $\Psi$ ← Backtrack($\Psi$,$\Pi$)
8:   **end for**
9: **end for**
10: **return** $\Psi$
```

Figure 8.2: Calculation of stabilizers of a marking.

8.4.2 Fewer Markings Canonicalized

A permutation symmetry specification is required to capture symmetries that are actually present in the CP-net. A permutation symmetry specification in accordance with the CP-net is said to be *consistent* [47]. The consistency of the permutation symmetry specification ensures [47] that for any two reachable markings, M and M', all binding elements (t, b) and all permutation symmetries $\phi \in \Phi_{SG}$: $M[(t,b)\rangle M' \Rightarrow \phi(M)[(t,\phi(b))\rangle\phi(M')$. This implies that for $\phi \in \Phi_{SG}^{M}$ we have that $M[(t,b)\rangle M' \Rightarrow M[(t,\phi(b))\rangle\phi(M')$. Hence two binding elements

```
1: Input M, Enabled(M)
2: $R \leftarrow$ Enabled(M)
3: for all $(t, b) \in R$ do
4:   for all $\phi \in \Phi_{SG}^{M} \cap$ CosetReps($\Phi_{SG}^{(t,b)}$) do
5:     if $(t, \phi(b)) \in$ R $\wedge$ $(\phi(b) \neq b)$ then
6:       $R \leftarrow R \setminus\{$(t,b)$\}$
7:     end if
8:   end for
9: end for
10: return $R$
```

Figure 8.3: Reducing binding elements.

(t, b_1) and (t, b_2) enabled in a marking M and satisfying that $(t, b_1) = (t, \phi(b_2))$ for some stabilizer ϕ of M will lead to symmetric markings. Hence only one of (t, b_1) and (t, b_2) needs to be considered for the construction of the condensed state space. Each binding element considered during construction of the condensed state space results in a calculation of the canonical representative for a marking. Hence, stabilizers can be used to reduce the number of calculations of canonical representatives.

We propose the algorithm Reduce in Fig. 8.3 for reducing the set of enabled binding elements considered in a marking M. Given a marking M and the set of enabled binding elements Enabled(M) in M Reduce calculates $R \subseteq$ Enabled(M) satisfying $(t, b_1), (t, b_2) \in R \wedge (t, b_1) \neq (t, b_2) \Rightarrow \forall \phi \in \Phi_{SG}^{M} : \phi(b_1) \neq b_2$. The algorithm introduces the extra cost of calculating the stabilizers of the marking M. However, this extra cost potentially results in fewer canonicalization of markings by reducing the number of successor states considered for M. Moreover, if we exploit the stabilizers when computing the canonical form as suggested in Sect. 8.4.1, then we have already computed the stabilizers for the marking as part of the canonicalization. It is worth observing that the algorithm Reduce in Fig. 8.3 for a given binding element (t, b) iterates $\phi \in \Phi_{SG}^{M} \cap$ CosetReps$(\Phi_{SG}^{(t,b)})$. Hence, Reduce exploits that it is only necessary to consider one representative from each of the cosets of the stabilizers of (t, b).

8.5 Exploiting Parallelism

We now give an algorithm which distributes the canonicalization of markings to a number of processes and in this way do canonicalization of markings in parallel. The basic idea behind this parallel algorithm is to use a number of *slave processes* $SLAVES = (sl_1, sl_2, \ldots, sl_m)$ for the time expensive canonicalization of markings, and a *master process* for construction of the condensed state space itself based on the canonical markings received from the slaves. The slave processes will implement the canonicalization of markings as described in the previous sections.

The algorithm used for state space construction is a modified version of the standard algorithm for condensed state space construction. The algorithm executed by the master process is shown in Fig. 8.4. The algorithm will be explained in detail below. The algorithm operates on a number of sets.

Unprocessed. The set of nodes/markings for which successors have not yet been calculated.

Waiting. The set of nodes/markings which have not yet been canonicalized.

BusySlaves. The set of slave processes which are currently busy, i.e., in the process of canonicalizing a marking.

Nodes. The set of states currently in the state space.

For the specification of the algorithm we have ignored that arcs are also often stored as part of the state space. It is however simple to modify the algorithm such that also the set of arcs in the state space is stored. The algorithm uses three communication primitives.

SEND (sl,M) which sends a marking M to a slave sl for canonicalization.

FINISHED (sl) which checks whether a slave sl has completed canonicalization of a marking.

RECEIVE (sl,M) which receives a canonicalized marking M from a slave sl.

Lines 1-4 initializes the four sets introduced above. The algorithm then consists of a loop which terminates if there are no more Unprocessed and Waiting markings, and there are no slaves which are in the process of canonicalizing a marking. The loop consists of three main parts. In the first part (lines 8-14) successor markings for all Unprocessed markings are calculated and added to the set of Waiting markings. In the second part (lines 16-25) it is checked whether any of the busy slaves have completed canonicalization (line 17). If so, the canonicalized marking is received (line 18) and if the marking is not already in the state space it is added to the set of Nodes and marked as Unprocessed (lines 19-22). The slave from which the marking was received is removed from the set of BusySlaves (line 23). Finally, in the third part (lines 27-33) markings from Waiting are sent to idle slaves for canonicalization. The ordering of the three main parts is deliberate. Having the calculation of successor markings first in the loop ensures that the state space generation proceeds in a width-first fashion which promotes the canonicalization to be done as much in parallel as possible. Checking which slaves have finished their job in the second part before distributing new markings to be canonicalized have the effect of distributing as many markings as possible in the third part. The algorithm can be modified to also exploit stabilizers to reduce the number of successor markings which have to be canonicalized as discussed in Sect. 8.4.2. This can be done by merging the first part of the algorithm with the second part so that the successors of markings received from the slaves are inserted immediately in the Waiting set without being added to the Unprocessed set. In this way Unprocessed need not be part of the algorithm.

```
Nodes $\leftarrow \{M_0\}$
Unprocessed $\leftarrow \{M_0\}$
Waiting $\leftarrow \emptyset$
BusySlaves $\leftarrow \emptyset$
**while** Unprocessed $\neq \emptyset \vee$ Waiting $\neq \emptyset \vee$
      BusySlaves $\neq \emptyset$ **do**
  {Calculate successor markings}
  **while** Unprocessed $\neq \emptyset$ **do**
    Select $M_1 \in$ Unprocessed
    Unprocessed $\leftarrow$ Unprocessed $- \{M_1\}$
    **for all** $((t,b),M_2)$ such that $M_1[(t,b)\rangle M_2$ **do**
      Waiting $\leftarrow$ Waiting $\cup \{M_2\}$
    **end for**
  **end while**
  {Retrieve markings from slaves which have finished}
  **for all** $sl_i \in$ BusySlaves **do**
    **if** FINISHED($sl_i$) **then**
      RECEIVE($sl_i$,$M_2$)
      **if** $M_2 \notin$ Nodes **then**
        Nodes $\leftarrow$ Nodes $\cup \{M_2\}$
        Unprocessed $\leftarrow$ Unprocessed $\cup \{M_2\}$
      **end if**
      BusySlaves $\leftarrow$ BusySlaves - $\{sl_i\}$
    **end if**
  **end for**
  {Distribute markings to idle slaves}
  **while** SLAVES - BusySlaves $\neq \emptyset \wedge$ Waiting $\neq \emptyset$ **do**
    Select $M_1 \in$ Waiting
    Select $sl_i \in$ SLAVES - BusySlaves
    Waiting $\leftarrow$ Waiting $- \{M_1\}$
    BusySlaves $\leftarrow$ BusySlaves $\cup \{sl_i\}$
    SEND($sl_i$,$M_1$)
  **end while**
**end while**
```

Figure 8.4: Parallel Canonicalization of Markings.

8.6 Experimental Results

A prototype have been developed implementing the algorithms presented in the previous sections. In this section we apply this prototype on a number of case-studies to give a first evaluation of the practicality of the proposed algorithms. In Sect. 8.6.1 we report on the experimental results obtained with the algorithms presented in Sect. 8.4. In Sect. 8.6.2 we report on the experimental results obtained with the parallel algorithm presented in Sect. 8.5.

The implementation is based on an integration between the DESIGN/CPN tool [26] which supports state space analysis of CP-nets, and the GAP tool

[34] which implements efficient representations and manipulations of algebraic groups. The Design/CPN state space tool implements the state space generation algorithm itself and the storage of the nodes and arcs of the state space. The algebraic techniques presented in Sect. 8.4 have been implemented in the GAP tool. The integration of the two tools operates as follows. Each time a new marking marking is generated during the condensed state space construction, it is sent to a GAP process that calculates the canonical representative. The representative marking computed is returned to Design/CPN which continues the condensed state space construction with the canonicalized marking received.

8.6.1 Stabilizer Algorithms

This section presents some experimental results obtained using the implementation of the techniques from Sect. 8.4. The results reported on in this section were obtained on a 333 MHz Pentium II Linux PC with 128 Mb of memory. The condensed state space has been constructed for a number of CPN models briefly described below. For a detailed description of the examples we refer to [47,59].

Distributed Database [47]. A CPN model of the communication between d symmetrical database managers.

Commit [59]. A CPN model of a two-phase commit protocol with a coordinator and w symmetrical workers.

Dining Philosophers [47]. A CPN model of the classical dining philosophers example with p philosophers. Only symmetries corresponding to rotations of the philosophers are considered.

The condensed state space has been generated for different configurations of the CPN models listed above. We study the generation time of the condensed state space using three different techniques.

All. Condensed state space generation with canonicalization of markings based on the algorithm in Fig. 8.1 which iterates through all the permutation symmetries allowed by the permutation symmetry specification. This technique is used for reference purposes.

Stabilizer. Condensed state space generation with canonicalization of markings based on considering one permutation symmetry from each of the cosets of the stabilizer groups (as described in Sect. 8.4.1).

Reduce. Condensed state space generation based on processing only one binding element from each equivalence class of the stabilizer group (as described in Sect. 8.4.2) combined with the canonicalization of markings as in Stabilizer.

Table 8.1 gives the generation time of the condensed state spaces using the three techniques presented above. The first four columns give information about

Table 8.1: Experimental Results – Stabilizer Algorithms.

CPN Model				All		Stabilizer		Reduce	
Name	Conf	$\|\Phi_{SG}\|$	Nodes	Canon	Time	Canon	Time	Canon	Time
Database	d=5	120	16	46	0:00:09	46	0:00:12	22	0:00:06
Database	d=6	720	22	77	0:00:43	77	0:00:28	32	0:00:14
Database	d=7	5,040	27	120	0:06:08	120	0:01:11	44	0:00:30
Database	d=8	40,320	37	–	–	177	0:03:00	58	0:01:08
Database	d=9	362,880	46	–	–	250	0:07:37	74	0:02:51
Commit	w=5	120	43	118	0:00:13	118	0:00:17	58	0:00:11
Commit	w=6	720	517	183	0:00:58	183	0:00:36	78	0:00:22
Commit	w=7	5,040	73	269	0:07:27	269	0:01:19	101	0:00:39
Commit	w=8	40,320	91	–	–	379	0:02:32	127	0:01:17
Commit	w=9	362,880	111	–	–	516	0:05:34	156	0:03:00
Dining phil.	p=16	16	143	1,270	0:01:24	1,270	0:01:41	1220	0:01:24
Dining phil.	p=17	17	211	1,990	0:02:27	1,990	0:02:50	1274	0:02:24

the CPN model. The Name column gives the name of the CPN model. The Conf column gives the configuration of the model, e.g., the number of database managers in the database example. The $|\Phi_{SG}|$ column gives the number of permutation symmetries allowed by the permutation symmetry specification. Finally, the Nodes column gives the number of nodes in the condensed state space. For each of the three techniques described above, the Canon column gives the number of markings canonicalized and the Time column gives the time it took to generate the condensed state space. The generation time is written on the form $hh : mm : ss$ where hh is hours, mm is minutes and ss is seconds. A – in an entry means that the condensed state space could not be generated with the available computing power. It should be noted that generation of condensed state spaces using the prototype implementation involves a certain amount of overhead resulting from the encoding and decoding of markings as well as the extra communication overhead introduced by the communication between DESIGN/CPN and GAP. The encoding/decoding of markings has not been optimized. Hence, the generation time should merely be seen as relative times which serve as a basis for comparing the three algorithms.

The generation algorithm used in Stabilizer is the same as the generation algorithm used in All. Hence, the number of markings canonicalized is the same using the two techniques. For the examples with only few permutation symmetries, e.g., the dining philosopher example where only rotations are considered, generation with All is faster than generation with Stabilizer. Simply to apply each of the permutations in turn is faster. From Table 8.1 it can, however, be seen that a significant speedup is obtained when the number of permutation symmetries allowed by the permutation symmetry specification increases, i.e., when the potential gain from only applying $|\Phi_{SG}|/|\Phi_{SG}^{M}|$ instead of $|\Phi_{SG}|$ permutation symmetries when canonicalizing a marking M increases.

The three columns Canon (one for each of the three techniques) show the number of markings canonicalized, i.e., the number of binding elements processed using All, Stabilizer and Reduce, respectively. When using All and Stabilizer all enabled binding elements are processed in each marking reached during generation. When using Reduce only binding elements belonging to different equivalence classes of the stabilizer group are considered. From Table 8.1 it can be seen that the number of binding elements processed using Reduce is much smaller than the number of binding elements processed using All and Stabilizer

Table 8.2: Experimental Results – Parallel Algorithm.

DBM	4	5	6	7	8	9	10
Nodes Full	109	406	1459	5104	17,497	59,049	196,831
Nodes Condensed	11	16	22	27	37	46	56
Slaves							
1	0:00:21	0:00:53	0:02:24	0:06:31	0:18:28	0:57:14	3:01:54
2	0:00:13	0:00:31	0:01:23	0:03:43	0:10:15	0:30:48	1:50:56
3	0:00:11	0:00:24	0:01:07	0:02:42	0:07:27	0:21:52	1:15:05
4	0:00:11	0:00:19	0:00:55	0:02:13	0:05:58	0:16:58	1:06:58
5	0:00:12	0:00:19	0:00:52	0:02:02	0:05:06	0:14:52	0:45:59
6	-	0:00:18	0:00:48	0:01:44	0:04:41	0:12:58	0:41:36
7	-	-	0:00:52	0:01:40	0:04:58	0:11:32	0:36:16
8	-	-	0:00:51	0:01:40	0:04:20	0:11:05	0:33:05
9	-	-	0:00:45	0:01:35	0:04:20	0:09:52	0:29:22
10	-	-	-	-	0:04:15	0:09:09	0:28:55
15	-	-	-	-	0:04:18	0:08:46	0:22:32
20	-	-	-	-	-	-	0:16:31
Speedup	1.75	2.94	3.20	4.12	4.29	6.52	11.01

(for 9 database managers 74 markings are canonicalized using Reduce compared to 250 markings when using Stabilizer). The reduction in the number of markings canonicalized is of course dependent of the CPN model in question. Furthermore, it can be seen that the time for generation of the condensed state space decreases when fewer markings are canonicalized. The prototype recalculates the stabilizers for the marking when reducing the binding elements. Hence, the numbers in Table 8.1 can potentially be further optimized if we exploit that these stabilizers have already been calculated as part of the canonicalization.

8.6.2 Parallel Algorithm

We now present some experimental results obtained with an implementation of the parallel algorithm from Sect. 8.5. We study the performance of the algorithm on the database example, and investigate the time for generation of the condensed state space for different numbers of database managers and slaves. The master process has been implemented in the DESIGN/CPN tool whereas the slaves have been implemented as a number of instances of the GAP tool running in parallel. The results reported on in this section were obtained with each of the slaves executing on a Sun Sparc4 Workstation each equipped with 64 Mb of memory. The master was running on a Sun ULTRA Enterprise 3000 with 512 Mb of memory. The slaves and the master were connected via a local area network.

Table 8.2 lists the generation time for the condensed state space for the number of database managers ranging from 4 to 10 and the number of slave processes ranging from 1 to 20. The DBM row lists the number of database managers. The Nodes Full row lists the number of states in the full state space for the given number of database managers. Similarly, the Nodes Condensed row lists the number of nodes in the condensed state space. The Slaves column lists the number of slaves, i.e., the number of GAP instances used for the generation of the condensed state space. A $-$ in an entry means that the condensed state space was not generated with that number of slaves. The reason for not considering that number of slaves was that using additional did not yield any significant speedup. The Speedup row gives the speedup in generation time obtained for a given number of database managers. The speedup has been calculated as the generation time with one slave divided by the generation time for the maximum number of slaves considered for that given number of database managers. It can be seen that the speedup obtained is significant in the beginning, becomes gradually smaller until a point is reached at which additional slaves results in only marginal or no reduction in generation time. It is also worth observing that the speedup increases with the number of database managers. The reason for this is that the canonicalization becomes more and more expensive since the number of permutation symmetries grows with the number of database managers, and because as the state space becomes large more nodes can be simultaneously in the waiting set.

8.7 Conclusions and Related Work

For large symmetry groups deciding whether two states are symmetric becomes time expensive due to the apparent high time complexity of the orbit problem. In this paper we have presented techniques to alleviate the negative impact of the orbit problem by the specification of canonical representatives for equivalence classes of states in CP-nets, and by giving algorithms exploiting stabilizers and parallelism for computing the condensed state space. Only the computation of the canonical representatives are specific to CP-nets. The results on the use of stabilizers and parallelism are valid also for other modelling formalisms where the symmetry method is applicable.

The specification, implementation, and experiments with the canonical form for CP-nets suggested in this paper is new. Previous work with condensed state spaces for symmetrical CP-nets, e.g., [53, 65], is based on a symmetry check between states, i.e., given two states s_1 and s_2 it is checked whether there is a permutation symmetry in Φ_{SG} mapping s_1 to s_2. The use of canonical forms is, however, attractive since one of the consequences is that, e.g., deterministic finite automata (DFAs) [40], binary decision diagrams (BDDs) [93], and graph encoded tuple sets (GETSs) [37] now can be used together with the symmetry method of CP-nets. Hence, the canonical forms can be used to speed up the the computation of condensed state spaces when the storage of the state space is based on explicit enumeration.

In [81] a method for canonicalizing states of Place/Transition nets is presented. The problem of iterating all symmetries is overcome by representing the symmetries as so called *generating set*. The method presented in [81], however, does not give perfect reduction for symmetry groups other than the groups of rotations and all permutations.

The use of cosets was originally suggested in [47], but no experimental results have been given before as to how effective they are in reducing the negative impact of the orbit problem. In [47] it was recognised that the use of stabilizers also can be used to reduce the number of binding elements that need to be considered for a given marking during calculation of the condensed state space, but no experimental results were given.

The suggestion to use the backtrack method for computing the set of stabilizers for markings was first presented in [51]. However, only its applicability on the database example was investigated in [51]. Moreover, is was based on a manual encoding of the states and did not consider the computation of the condensed state space. The results were only concerned with computing the stabilizers for provided markings – it was not done integrated with the state space generation. One reason for this was that there were no integration between the DESIGN/CPN tool and the GAP tool. The development of this integration is another contribution of the work presented in this paper.

The aspect of parallelization of state space generation has also been studied in [86] which describes parallelization of the Murϕ verifier. In our work we parallelize the time expensive canonicalization of markings, whereas the enabling calculation and the storage of the condensed state space is located within the master process. In [86] the enabling calculation as well as the storage of the state space are distributed among a number of processes. Our experimental results presented in Sect. 8.6.2 are promising with respect to the generation time of the condensed state spaces. Furthermore, with our technique we can still use all standard model checking algorithms because the state space is stored at a single site.

Chapter 9

Coloured Petri Nets and State Space Generation with the Symmetry Method

The paper *Coloured Petri Nets and State Space Generation with the Symmetry Method* constituting this chapter has been published as a workshop paper [63].

[63] L. Lorentsen. Coloured Petri Nets and State Space Generation with the Symmetry Method. To appear in *Proceedings of the 4th Workshop on Practical use of Coloured Perti Nets and CPN/Tools*, Department of Computer Science, University of Aarhus, Denmark, august 2002.

The contents of this chapter is equal to the workshop paper [63] except for minor typographical changes.

Coloured Petri Nets and State Space Generation with the Symmetry Method

Louise Lorentsen*

Abstract

This paper discusses state space generation with the symmetry method in the context of Coloured Petri Nets (CP-nets). The paper presents the development of the Design/CPN OPS tool which, together with the Design/CPN OE/OS tool, provides fully automatic generation of symmetry reduced state spaces for CP-nets with consistent symmetry specifications. Practical experiments show that the practical applicability of the symmetry method is highly depended on efficient algorithms for determining whether two states are symmetric. We present two techniques to obtain an efficient symmetry check between markings of CP-nets: a technique that improves the generation time and a technique that reduces the memory required to handle the symmetries during calculation. The presented algorithms are implemented in the Design/CPN OPS tool and their applicability is evaluated based on practical experiments.

9.1 Introduction

The state space of a system is a directed graph with a node for each reachable state of the system and an arc for each state change. From the state space it is possible to verify whether the system possesses a set of desired properties, e.g., the absence of deadlocks, the possibility to always reenter the system's initial state, etc. However, the practical use of state spaces for formal analysis and verification of systems is often limited by the *state explosion problem* [90]: even small systems may have a large (or infinite) number of states, thus making it impossible to construct the full state space of the system. Several reduction techniques have been suggested to alleviate the state explosion problem. An example of such a reduction technique is the *symmetry method* [24, 28, 44, 48]. The symmetry method is not restricted to a specific modelling language. In this paper we work with the symmetry method for Coloured Petri Nets (CP-nets or CPN) [47, 48]. The basic observation behind the symmetry method is that many concurrent and distributed systems posses a degree of symmetry which is also reflected in the state space. The idea behind the symmetry method is to factor out this symmetry and obtain a *condensed state space* which typically

*Department of Computer Science, University of Aarhus, Aabogade 34, DK-8200 Aarhus N. DENMARK, E-mail: `louisel@daimi.au.dk`.

is much smaller than the full state space, but from which the same properties can be verified without unfolding to the full state space. The symmetries in such systems can be described by algebraic groups of permutations. For CP-nets the symmetries used for the reduction are induced by algebraic groups of permutations on the atomic colour sets of the CP-net. Hence, we will also use the term *state spaces with permutation symmetries* (SSPSs) to denote the condensed state spaces obtained by using the symmetry method.

In the context of CP-nets the theory of the symmetry method is well developed [47, 48] and a computer tool (the Design/CPN OE/OS tool [52, 53]) that supports state space generation with the symmetry method has been developed. However, the symmetry method in the context of CP-nets has only few applications in practice, e.g. [31, 65]. One of the drawbacks of the Design/CPN OE/OS tool is that it requires the user to implement two predicates determining whether two states/actions are symmetric or not. This requires both programming skills and a deep knowledge of the symmetry method. This is especially the case if the predicates are required to be efficient. However, when constructing SSPSs for CP-nets, it can be observed that the predicates can be automatically deduced [47] provided the algebraic groups of permutations used for the reduction have been specified[1]. The above observation has motivated the construction of a tool (the Design/CPN OPS tool [64]) which, given an assignment of algebraic groups of permutations to the atomic colour sets of the CPN model, generates the predicates for the Design/CPN OE/OS tool expressing whether two states/actions are symmetric.

The problem of determining whether two states/actions are symmetric is in literature also referred to as *the orbit problem* and an efficient solution to this problem is a central issue for the applicability of the symmetry method. The computational complexity of the orbit problem has been investigated in [24] showing that in general it is at least as hard as *the graph isomorphism problem* for which no polynomial time algorithm is known. However, in the context of CP-nets symmetry can be determined efficiently in a number of special cases, e.g., when the CP-net only contains atomic colour sets [4, 47]. This is, however, not true in general; the problem of determining symmetry between states/actions is complicated by the fact that colour sets can contain arbitrary structural dependencies.

During development of the Design/CPN OPS tool a number of practical experiments have been performed with different strategies for the implementation of the predicates. The practical experiments show that the chosen strategy for the implementation of the predicates greatly influences whether the symmetry method for CP-nets is applicable in practice. The algebraic groups of permutations used for the reduction potentially becomes very large as the system parameters grow. The number of symmetries used for the reduction is potentially $\prod_{A \in \Sigma_A} |A|!$ where Σ_A denotes the atomic colour sets of the CPN model.

[1]There are two main approaches in the literature: either the permutations can be automatically deduced from the model, e.g., [14, 80], or explicitly specified by the modeller [47, 48]. The latter approach is based on the belief that the modeller, who constructs the model is familiar with the system modelled and has an intuitive idea of the symmetries present in the model [47].

Hence, efficient handling of the symmetries used for the reduction becomes an important aspect when developing algorithms for the predicates used in the symmetry method.

In this paper we present techniques and algorithms which implements an efficient solution to the orbit problem in SSPS generation for CP-nets. The algorithms presented in this paper are based on general techniques which can be applied independently of model specific details. Hence, the predicates can be automatically constructed by the tool.

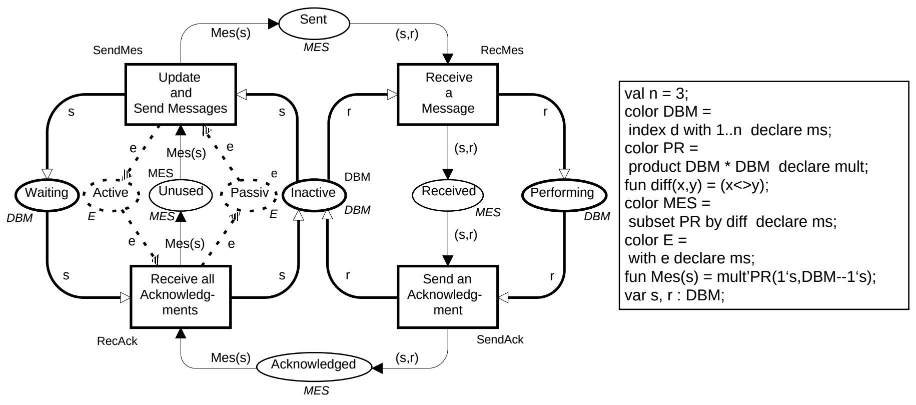
The paper is structured as follows. Section 9.2 presents the symmetry method for CP-nets by means of an example. Section 9.3 presents the basic generation algorithm for SSPSs. Section 9.4 introduces a basic solution to the orbit problem for CP-nets that will be used for reference purposes. Section 9.5 presents algorithms that improve the run-time of the basic algorithm. Section 9.6 presents algorithms that ensure an compact representation of the symmetries throughout calculation of the SSPSs. Finally, Sect. 10.6 contains the conclusions.

9.2 The Symmetry Method for CP-nets

In this section we introduce the symmetry method for CP-nets by means of an example. We will use the example of a distributed database from Sect. 1.3 in [46]. Section 9.2.1 presents the CPN model of the distributed database and show how the symmetry method can be used to reduce the size of the state space. Section 9.2.2 explains how the symmetries used in the symmetry method are specified as permutations of atomic colours. Finally, Sect. 9.2.3 presents a data structure which can be used to represent sets of symmetries in a CP-net.

9.2.1 Example: Distributed Database

The CP-net for the distributed database is shown in Fig. 9.1. The CP-net models a simple distributed database with n different sites. Each site contains a copy of all data and this copy is handled by a database manager. Each



```
val n = 3;
color DBM =
 index d with 1..n  declare ms;
color PR =
 product DBM * DBM  declare mult;
fun diff(x,y) = (x<>y);
color MES =
 subset PR by diff  declare ms;
color E =
 with e declare ms;
fun Mes(s) = mult'PR(1's,DBM--1's);
var s, r : DBM;
```

Figure 9.1: CP-net for the Distributed Database example.

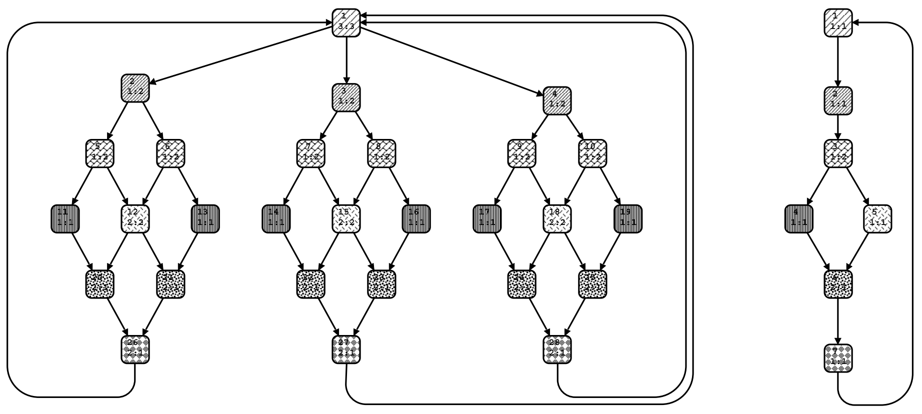


Figure 9.2: The full state space (the left-hand side) and SSPS (the right-hand side) of the CP-net for the distributed database example with 3 symmetric database managers($n = 3$).

database manager can change its own copy of the database and send a message to all other database managers requesting them to update their copy of the database. The distributed database system uses the indexed colour set `DBM` to model the database managers, the enumeration colour set `E` to model whether the protocol is active, and the product colour set `MES` to model the messages. The content of the database and the messages are not modelled. Only header information (the sender and the receiver) is contained in a message.

The distributed database system possesses a degree of symmetry. The database managers are treated similarly, only their identities differ. This symmetry is also reflected in the state space of the distributed database system. The state space for the CPN model with three database managers is shown in the left-hand side of Fig. 9.2. The idea behind SSPSs is to factor out this symmetry and obtain a smaller state space from which the properties of the distributed database system can be verified without unfolding to the full state space. When constructing the SSPS for the database system we consider two markings/binding elements to be symmetric if they are equal except for a bijective renaming of the database managers. This kind of symmetry (based on bijective renamings) induces two equivalence relations; one on the set of markings and one on the set of binding elements [47]. The basic idea when constructing the SSPS is to lump together symmetric markings/binding elements into one node/arc, i.e., only store one representative from each equivalence class. The right-hand side of Fig. 9.2 shows the SSPS for the distributed database. The nodes in the full state space (in the left-hand side of the figure) are coloured such that nodes corresponding to symmetric markings have the same colour. The same colours are used in the SSPS (in the right hand side of the figure). From the figure it can be seen that the SSPS only contains one node per equivalence class of symmetric markings.

9.2.2 Symmetry Specification

The symmetries used for the reduction are obtained from permutations of the atomic colours in the CPN model. Let Σ_A denote the set of atomic colour sets of the CPN model. For each atomic colour set in the CPN model, $A \in \Sigma_A$, we define an algebraic group of permutations Φ_A, i.e., a subgroup of $[A \to A]$. A symmetry ϕ of the system is a set of permutations of the atomic colour sets of the model, i.e., $\phi = \{\phi_A \in \Phi_A\}_{A \in \Sigma_A}$. In the rest of the paper we will use the term *permutation symmetry* to denote a set of permutations of the atomic colour sets of a CPN model.

The symmetry considered in the distributed database system is a bijective renaming of the database managers. This is obtained by allowing all permutations of the atomic colour set `DBM`. Hence a permutation symmetry in the distributed database system is a set $\phi = \{\phi_E \in \Phi_E, \phi_{DBM} \in \Phi_{DBM}\}$, where $\Phi_{DBM} = [\text{DBM} \to \text{DBM}]$ and $\Phi_E = \{\phi_{id}\}$ (where ϕ_{id} is the identity permutation, i.e., $\phi_{id}(\mathtt{e})=\mathtt{e}$). From the permutation symmetries of the CPN model we derive permutations of the structured colour sets, multi-sets, markings and binding elements as described in [47].

A *symmetry specification* of a CP-net is an assignment of algebraic groups of permutations to each of the atomic colour sets of the CP-net and hence determines a group of permutation symmetries. The symmetry specification is required to be consistent [47] which means that it is required to only express symmetries that are actually present in the system. We will use Φ_{SG} to denote the group of permutation symmetries given by a consistent symmetry specification SG. In the rest of the paper we assume that a CP-net with places $P = \{p_1, p_2, ..., p_n\}$ is given together with a consistent symmetry specification SG which determines a group Φ_{SG} of permutation symmetries.

9.2.3 Restriction Sets

A consistent symmetry specification SG determines a group of permutation symmetries Φ_{SG}. During generation of the SSPS we need some kind of representation of Φ_{SG}. One possibility is to list the permutation symmetries. Since the symmetry groups used for the reduction can be very large, this is not a feasible solution.

A set of permutations of an atomic colour set can instead be represented as a *restriction set*. Restriction sets are introduced in [47] and formally defined in [4]. Here we will introduce restriction sets by means of an example. Below we use a restriction set to represent a subset of [`DBM` $\to$ `DBM`]. The set of permutations mapping `d(1)` to `d(2)` and the set {`d(2)`,`d(3)`} to the set {`d(1)`,`d(3)`} can be represented by the following restriction set:

`d(1)`	`d(2)`
`d(2) d(3)`	`d(1) d(3)`

Each row in the restriction set introduces a requirement for the set of permutations represented by the restriction set. The individual restrictions (rows) express that the colours on the left-hand side must be mapped into the colours of the right-hand side. In [4] it is proven that restriction sets can be efficiently intersected (while maintaining the compact representation) and that an arbitrary set of permutations can be represented by a set of restriction sets. Hence, restriction sets provide a potentially compact representation of sets of permutations. In the rest of the paper we will use restriction sets to represent sets of permutations of atomic colour sets. Hence, a symmetry specification can be represented by a set of restriction sets for each atomic colour set in the CP-net.

9.3 Condensed State Space Generation

In this section we give an introduction to the standard algorithm GENERATESSPS for construction of the SSPS of a CP-net [47]. *Nodes* and *Arcs* are sets of states (markings) and actions (binding elements), respectively, and it contains the states and actions that are included in the SSPS. *Unprocessed* is a set of states, and contains the states for which we have not yet calculated the successor states. M_0 denotes the initial state. NEXT(M) is a function calculating the set of possible next moves (an action and the resulting state) from the state M. NODE(M) is a function that checks whether a node symmetric to M is already included in the SSPS. If not, M is added to *Nodes* and *Unprocessed*. Similarly, ARC(M_1,b,M_2) is a function that checks whether a symmetric arc is already included in the SSPS, i.e., an arc consisting of a binding element symmetric to b from a marking symmetric to M_1 to a marking symmetric to M_2. If not, (M_1,b,M_2) is added to *Arcs*.

Algorithm: GENERATESSPS () =

```
1:  $Nodes \leftarrow \{M_0\}$
2:  $Arcs \leftarrow \emptyset$
3:  $Unprocessed \leftarrow \{M_0\}$
4:  repeat
5:     select $M_1 \in Unprocessed$
6:     for all $(b,M_2) \in \mathsf{Next}(M_1)$ do
7:        NODE($M_2$)
8:        ARC($M_1$,$b$,$M_2$)
9:     end for
10:    $Unprocessed := Unprocessed \setminus \{M_1\}$
11: until $Unprocessed = \emptyset$
```

The algorithm proceeds in a number of iterations. In each iteration a state (M_1) is selected from *Unprocessed* and the successor states (and actions) are calculated using the NEXT function. For each of the successor states, M_2, it is checked whether *Nodes* already contains a state symmetric to M_2. If not M_2 is added to both *Nodes* and *Unprocessed*. Similar checks are made for the actions. The check for symmetric states and symmetric actions are instances of the *orbit problem*. From the basic generation algorithm it can be seen that efficient generation of the SSPS is highly dependent on the efficiency of the algorithms for determining the following two problems: 1) When reaching a new marking

M during generation of the SSPS, is there a marking symmetric to M already included in the SSPS? And 2) When reaching a new arc (M_1, b, M_2) during generation of the SSPS, is there a symmetric arc already included in the SSPS?

GENERATESSPS is implemented in the Design/CPN OE/OS tool and used when calculating SSPSs for CP-nets. In Design/CPN a hash function is used when storing the markings during generation of the SSPS. When reaching a new marking during generation of the SSPS each marking stored with the same hash value is checked to see if it is symmetric to the newly reached marking, i.e., symmetry checks are performed locally between markings in the collision lists. The user of the tool is free to use his own hash function. The only requirement is that the hash function used is symmetry respecting, i.e., symmetric states are mapped to the same hash value. This is the case for the default hash function used in Design/CPN. Hence, when using the Design/CPN OE/OS tool for the generation of SSPSs efficient generation is dependent on the efficiency of the two predicates, P_M and P_{BE}, determining symmetry between markings and binding elements, respectively.

P_M: Given $M_1, M_2 \in \mathbb{M}$ determine whether $\exists \phi \in \Phi$ s.t. $\phi(M_1) = M_2$.

P_{BE}: Given $(t_1, b_1), (t_2, b_2) \in \mathrm{BE}$ determine whether $\exists \phi \in \Phi$ s.t. $\phi(t_1, b_1) = (t_2, b_2)$.

The Design/CPN OE/OS tool requires P_M and P_{BE} to be implemented by the user. Implementing such predicates is error-prone for large CPN models and requires both programming skills and a detailed knowledge of the symmetry method. This is especially the case if the predicates are required to be efficient. The required user implementation of P_M and P_{BE} in the Design/CPN OE/OS tool has motivated the development of the Design/CPN OPS tool which given a CP-net and a consistent symmetry specification automatically generates the two predicates, P_M and P_{BE}, needed by the Design/CPN OE/OS tool. In the rest of the paper we will present techniques and algorithms to obtain implementations of P_M and P_{BE} in the Design/CPN OPS tool which are efficient in practice. The algorithms are independent of the specific CP-net. Hence, the predicates can be automatically generated.

In the following discussions we will concentrate on the markings since the symmetry check between binding elements can be viewed as a special case of symmetry checks between markings: Given a transition t with variables $v_1, v_2, \ldots, v_m$, a binding b of t can be viewed as a vector of singleton multi-sets $(1`b(v_1), 1`b(v_2), \ldots, 1`b(v_m))$, where $b(v)$ denotes the value assigned to v in the binding b. Since transitions cannot be permuted by permutation symmetries in CP-nets determining symmetry between binding elements is the same as determining symmetry between markings. Hence, in the rest of the paper we will present techniques and algorithms to obtain an efficient implementation of P_M, i.e., given $M_1, M_2 \in \mathbb{M}$ determine whether $\exists \phi \in \Phi$ such that $\phi_{\mathbb{M}}(M_2) = M_1$.

9.4 Basic Algorithm for P_M

In this section we will present a basic algorithm which implements the predicate P_M. Section 9.4.1 presents the algorithm. Section 9.4.2 presents experimental results obtained using the Design/CPN OPS tool where the basic algorithm presented in this section is used to determine symmetry between markings.

9.4.1 Presentation of the Algorithm

The algorithm is based on a simple approach where Φ_{SG}, i.e., the group of permutation symmetries allowed by the symmetry specification SG, is iterated to determine whether $\exists \phi \in \Phi_{SG}$ s.t. $\phi(M_1) = M_2$. The algorithm P_M^{Basic} is given below.

Algorithm: $P_M^{Basic}(M_1, M_2)$

```
1: for all $\phi \in \Phi_{SG}$ do
2:   if $\phi(M_1) = M_2$ then
3:     return true
4:   end if
5: end for
6: return false
```

The algorithm repeatedly selects a permutation symmetry ϕ from Φ_{SG} (line 1) and tests whether ϕ is a symmetry between the two markings, M_1 and M_2 given as input (lines 2-4). The iteration stops when a permutation symmetry ϕ for which $\phi(M_1) = M_2$ is found (line 3) or the entire Φ_{SG} has been iterated (line 6).

The algorithm P_M^{Basic} potentially tests fewer permutation symmetries than $|\Phi_{SG}|$. This is however not the case if M_1 and M_2 are not symmetric. In that case the algorithm checks the whole Φ_{SG}. Hence, P_M^{Basic} is only useful for CP-nets with few permutation symmetries. This is also supported by the experimental results presented below.

However, before we present the experimental results of the P_M^{Basic} algorithm we will briefly introduce how it is tested whether a permutation symmetry ϕ maps a marking M_1 to another marking M_2 (line 2 in P_M^{Basic}). In [4] it is shown how the set of permutation symmetries between two markings can be determined as the intersection of the sets of permutation symmetries between the markings of the individual places. Hence, to determine whether a permutation symmetry $\phi \in \Phi_{SG}$ is a symmetry between two markings M_1 and M_2, we in turn test the multi-sets constituting the markings of $p_i \in P$. Note that if a permutation symmetry is not a symmetry for the marking of a place $p_i \in P$, i.e., $\phi(M_1(p_i)) \neq M_2(p_i)$ the permutation symmetry ϕ cannot be a symmetry between M_1 and M_2 and therefore there is no need to test the remaining places in P. Using the ideas presented in [4] we obtain an algorithm TestPermutationSymmetry which given a permutation symmetry ϕ and two markings,

M_1 and M_2, tests whether $\phi(M_1) = M_2$.

```
Algorithm: TestPermutationSymmetry($\phi$,$M_1$,$M_2$) =
1: for all $p_i \in P$ do
2:   if $\phi(M_1(p_i)) \neq M_2(p_i)$ then
3:     return false
4:   end if
5: end for
6: return true
```

The algorithm repeatedly selects a place $p_i \in P$ of the CP-net (line 1) and tests whether ϕ is a symmetry between the markings of p_i in M_1 and M_2 (line 2-4). If not, ϕ is not a symmetry between M_1 and M_2, otherwise a new place is tested. The iteration proceeds until a place $p_i \in P$ for which ϕ is not a symmetry is found (line 3) or all places have been tested (line 6).

9.4.2 Experimental Results of the P_M^{Basic} Algorithm

This section presents experimental results obtained using the Design/CPN OPS tool. The following results are obtained using an implementation of P_M^{Basic} to determine whether two markings are symmetric. A similar approach is used for the implementation of P_{BE}.

The Design/CPN OPS tool represents Φ_{SG} as a restriction set. When checking symmetry between two markings using P_M^{Basic} Φ_{SG} is listed and the permutation symmetries from the list are removed and tested until a permutation symmetry ϕ is found for which $\phi(M_1) = M_2$ or the list is empty.

SSPSs have been generated for two different CP-nets in a number of configurations. The CP-nets used in the experiments are briefly described below. For a detailed description of the CP-nets we refer to [47, 59].

Commit [59]. A CP-net modelling a two-phase commit protocol with a coordinator and w symmetrical workers.

Distributed database [47]. The CP-net presented in Sect. 9.2 modelling the communication between n symmetrical database managers.

Table 9.1 shows the generation statistics for of the SSPS for different configurations of the two CP-nets using the P_M^{Basic} algorithm. The CP-net column gives the name (C stands for commit and D stands for distributed database) and configuration of the CP-net for which the SSPS is generated as well as the number of permutation symmetries given by the symmetry specification SG used for the reduction. The Count column gives two numbers: the total number of times the P_M predicate is called during calculation of the SSPS and the number of calls which evaluate to true, i.e., the number of those calls which determine that the two markings are symmetric. The Tests column presents statistics on the number of permutation symmetries applied to markings during generation of the SSPS: Total gives the total number of permutation symmetries applied

				P_M^{Basic}				
CP-net		Count		Tests				Time
Con.	$\|\Phi_{SG}\|$	P_M	P_M=`true`	Total	P_M^{Basic}	P_M^{Basic} =`true`	% $\|\Phi_{SG}\|$	Secs
C_2	2	11	7	19	1.73	1.57	78.5	0
C_3	6	26	19	90	3.46	2.53	42.0	0
C_4	24	53	41	488	9.21	4.88	20.3	0
C_5	120	95	76	3,242	34.1	12.7	10.5	0
C_6	720	157	127	27,297	174	44.86	6.2	23
C_7	5,040	–	–	–	–	–	–	–
D_2	2	4	2	7	1.75	1.50	75.0	0
D_3	6	14	6	61	4.36	2.17	36.0	0
D_4	24	35	15	533	15.2	3.53	14.7	0
D_5	120	71	31	5,037	70.9	7.64	6.37	0
D_6	720	126	56	51,693	410	23.1	3.20	16
D_7	5,040	–	–	–	–	–	–	–

Table 9.1: Generation statistics for SSPS generation using the P_M^{Basic} algorithm.

to markings during generation of the SSPS, P_M^{Basic} gives the average number of permutation symmetries applied in each call of P_M^{Basic}, P_M^{Basic}=`true` gives the average number of permutation symmetries applied in each call of P_M^{Basic} which evaluates to true (the case where iteration of the entire Φ_{SG} is potentially avoided), and finally, % $|\Phi_{SG}|$ gives the average percentage of the permutation symmetries which are tested in a call of P_M^{Basic}. Finally, the Time column gives the number of seconds it took to generate the SSPS for the given CP-net. A '–' in an entry means that the SSPS could not be generated within 600 seconds. All experimental results presented in this paper are obtained on a 333MHz PentiumII PC running Linux. The machine is equipped with 128 Mb RAM.

From Table 9.1 it can be seen that when system parameters increase the number of permutation symmetries tested increase significantly. This is caused by the increasing size of Φ_{SG}. From the % $|\Phi_{SG}|$ column it can be seen that the average percentage of Φ_{SG} which are tested in P_M^{Basic} decreases when the system parameters increase. However, the increasing size of Φ_{SG} makes it impossible to generate the SSPS for the two CP-nets when system parameters, i.e., the number of concurrent readers or database managers, becomes greater than 6. This is also caused by the approach where Φ_{SG} is listed before the permutation symmetries are tested. For systems of increasing size $|\Phi_{SG}|$ imply that the entire Φ_{SG} cannot be represented in memory and, thus, generation of the SSPS is not possible. It should be noted that the results presented in Table 9.1 depends on the order in which the permutation symmetries are applied. The order used for the experiments is the same order in each call of P_M^{Basic} based on a recursive unfolding of the restriction set.

We conclude that the experiments performed using P_M^{Basic} in generation of SSPSs show that the run-time incurred by the iteration of Φ_{SG} becomes significant when system parameters grow. Hence, in order to make the calculation of SSPSs for CP-nets applicable in practice we need to carefully consider the number of permutation symmetries tested in the generation of the SSPSs. The next section presents techniques which improve P_M^{Basic} in this direction.

9.5 Approximation Techniques

In this section we will present an algorithm which presents an improved implementation of the predicate P_M^{Basic}. Section 9.5.1 presents the algorithm. Section 9.5.2 presents experimental results obtained using the Design/CPN OPS tool where the improved algorithm presented in this section is used to determine symmetry between markings.

9.5.1 Presentation of the Algorithm

The problem when using P_M^{Basic} for the symmetry check between markings is that in the worst case $|\Phi_{SG}|$ permutation symmetries will be checked. When determining symmetry between markings a selection of simple checks can in many cases determine that two markings are not symmetric or determine a smaller set of permutation symmetries that have to be checked.

In this section we will present a new algorithm for P_M which given two markings, M_1 and M_2, calculates a set Ψ_{M_1,M_2} such that $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\} \subseteq \Psi_{M_1,M_2} \subseteq \Phi_{SG}$. Hence, Ψ_{M_1,M_2} is a super-set of the set of permutation symmetries mapping M_1 to M_2. If $\Psi_{M_1,M_2} = \emptyset$ we can conclude that M_1 and M_2 are not symmetric. However, if Ψ_{M_1,M_2} is non-empty we have to test the individual permutation symmetries in Ψ_{M_1,M_2}. In worst case $|\Psi_{M_1,M_2}|$ permutation symmetries have to be checked. This is the case if M_1 and M_2 are not symmetric. If M_1 and M_2 are symmetric then in worst case $|\Psi_{M_1,M_2}| - |\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}| + 1$ permutation symmetries have to be checked. Hence, the goal of the approximation technique is to construct Ψ_{M_1,M_2} as close to $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$ as possible.

In [4] it was shown that if a CP-net only contains atomic colour sets then the set $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$ can be determined efficiently. This is, however, not the case if the CP-net contains structured colour sets. Nevertheless, we will use the technique to efficiently obtain an approximation Ψ_{M_1,M_2} of $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$, thus reducing the number of permutation symmetries which have to be checked compared to the approach used in P_M^{Basic}. This is obtained at the cost of doing the approximation. In the following we will show how such an approximation can be obtained efficiently when Φ_{SG} is represented as a restriction set. The approximation technique is based on ideas from [4,47].

The set of permutation symmetries mapping a marking M_1 to another marking M_2 can be found as the intersection of sets of permutation symmetries mapping $M_1(p_i)$ to $M_2(p_i)$ for all $p_i \in P$. Similarly, it is shown in [47] and proved in [4] how the set of permutation symmetries between such markings of places, i.e., multi-sets, can be determined as the intersection over sets of symmetries between sets with equal coefficient in the multi-sets, i.e., for a permutation symmetry to be a symmetry between ms_1 and ms_2 it must ensure that a colour appearing with coefficient c in ms_1 must be mapped into a colour appearing with the same coefficient in ms_2. We will illustrate using the CP-net of the Distributed Database (Fig. 9.1) as an example.

Let ms_1=`1‘d(2)+1‘d(3)` and ms_2=`1‘d(1)+1‘d(2)` be two markings of a the place `Inactive` with colour set `DBM`. In ms_1 two colours (`d(2)` and `d(3)`) appear with coefficient 1 and one colour (`d(1)`) appear with coefficient 0. We can express the multi-set of coefficients as `2‘1+1‘0`. In ms_2 it is also the case that two colours (`d(1)` and `d(2)`) appear with coefficient 1 and one colour (`d(3)`) appear with coefficient 0. Hence, ms_2 has the same multi-set of coefficients as ms_1 namely `2‘1+1‘0`. A permutation ϕ_{DBM} of the colour set `DBM` is a permutation mapping ms_1 to ms_2 if ϕ_{DBM} ensures that a colour appearing with coefficient 1 in ms_1 is mapped to a colour appearing with coefficient 1 in ms_2, and similar for the rest of the coefficients (here just 0). Hence, we can construct a restriction set representing the set of permutations between ms_1 and ms_2 by constructing a restriction for each of the coefficients appearing in ms_1 and ms_2.

Coefficient 0:	`d(1)`	`d(3)`
Coefficient 1:	`d(2) d(3)`	`d(1) d(2)`

In the above example the two multi-sets had the same multi-sets of coefficients. This is a necessary requirement for the two multi-sets to be symmetric [4]. If not, the left and right-hand sides of the constructed restrictions do not contain the same number of elements, and thus does not represent a valid set of permutations. Multi-sets of coefficients are formally defined in [4]. We define multi-sets of coefficients using the notation used in this paper below and present an algorithm which calculates the set of permutation symmetries between two multi-sets over an atomic colour set.

Definition 9.1

For a multi-set ms over a colour set C we define COEFFICIENTS$_C$*(ms) as the set of coefficients appearing in ms:*

$$\text{COEFFICIENTS}_C(ms) = \{i \in \mathbb{N} | \exists c \in C \textit{ such that } ms(c) = i\}$$

Let ms be a multi-set over a colour C. For $i \in$ COEFFICIENTS$_C(ms)$ we define the i-coefficient-class for ms as the set of colours in C appearing with coefficient i:

$$\mathsf{C}_i(ms) = \{c \in C | ms(c) = i\}$$

We define the multi-set of coefficients for ms by

$$\text{CFMS}(ms) = \{ms(i)\text{‘}i\}_{i \in \mathsf{Coefficients}_C(ms)}$$

Based on the above definitions we formulate an algorithm FINDPERMUTATIONS which given two multi-sets ms_1 and ms_2 over an atomic colour set $A \in \Sigma_A$ calculate the set $\{\phi_A \in \Phi_A \mid \phi_A(ms_1) = ms_2\}$.

Algorithm: FINDPERMUTATIONS$_{ms}$(ms_1,$ms2$)

```
1: if $\text{CFMS}(ms_1) = \text{CFMS}(ms_2)$ then
2:     return $\{(C_i(ms_1), C_i(ms_2))\}_{i \in \mathsf{Coefficients}(ms_1)}$
3: else
```

```
4:     return ∅
5: end if
```

The algorithm tests whether $\textsc{Cfms}(ms_1) = \textsc{Cfms}(ms_2)$ (line 1), i.e., the multi-set of coefficients are equal. If not ms_1 and ms_2 are not symmetric [4], i.e., the empty set is returned (line 4), otherwise a restriction set is constructed containing a restriction $(C_i(ms_1),C_i(ms_2))$ for each of the coefficients i in $\textsc{Coefficients}(ms_1)$ (line 2).

Given two markings, M_1 and M_2, the algorithm $\textsc{FindPermutationSymmetries}_M$ calculates the a set of permutation symmetries Ψ_{M_1,M_2} as the intersection of Φ_{SG} and the sets of permutations between the markings of the individual places with atomic colour sets (calculated using $\textsc{FindPermutations}_{ms}$).

Algorithm: $\textsc{FindPermutationSymmetries}_M(M_1,M_2) =$

```
1: $\Phi' \leftarrow \Phi_{SG}$
2: for all $p \in \{p' \in P \mid p'$ has an atomic colour set$\}$ do
3:     $\Phi' \leftarrow \Phi' \cap \textsc{FindPermutations}_{ms}(M_1(p),M_2(p))$
4: end for
5: return $\Phi'$
```

If the CP-net only contains places with atomic colour sets the set Ψ_{M_1,M_2} of permutation symmetries calculated using $\textsc{FindPermutationSymmetries}_M$ is equal to the set $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$. If the CP-net also contains places with structured colour sets then Ψ_{M_1,M_2} is a super-set of $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$, i.e, $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\} \subseteq \Psi_{M_1,M_2}$. We will use $\textsc{FindPermutationSymmetries}_M$ to improve the P_M^{Basic} algorithm presented in Sect. 9.4, i.e., to reduce the number of permutation symmetries which have to be checked. The new algorithm P_M^{Approx} is presented below.

Algorithm: $P_M^{Approx}(M_1, M_2)$

```
1: for all $\phi \in \textsc{FindPermutationSymmetries}_M\ (M_1,M_2)$ do
2:     if TestPermutationSymmetry' $(\phi,M_1,M_2)$ then
3:         return true
4:     end if
5: end for
6: return false
```

The algorithm repeatedly selects a permutation symmetry ϕ from the set of permutation symmetries approximated using $\textsc{FindPermutationSymmetries}_M$ (line 1) and tests whether ϕ is a symmetry between the two markings (lines 2-4). The iteration stops when a permutation symmetry ϕ for which $\phi(M_1) = M_2$ is found (line 3) or the entire set has been iterated (line 6). P_M^{Approx} (M_1, M_2) uses TestPermutationSymmetry' (ϕ,M_1,M_2), a modified version of the algorithm TestPermutationSymmetry presented in Sect. 9.4, to determine whether $\phi(M_1) = M_2$. The difference is that given a permutation symmetry ϕ

and two markings, M_1 and M_2, TESTPERMUTATIONSYMMETRY' only test ϕ on the places which have a structured colour set. The markings of the places with atomic colour sets are already accounted for in the approximation and do not have to be tested again.

The complexity of the calculation of FINDPERMUTATIONSYMMETRIES$_M$ is independent of $|\Phi_{SG}|$. This is a very attractive property, since the experimental results presented in Sect. 9.4 showed that iterating the group of permutation symmetries is not applicable in practice when the symmetry specification determines a large set of permutation symmetries.

If the CP-net contains places with atomic colour sets P_M^{Approx} potentially tests fewer permutation symmetries than P_M^{Basic}. In P_M^{Basic} at most $|\Phi_{SG}| - |\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}| + 1$ permutation symmetries are checked when determining whether two markings are symmetric, whereas at most $|$FINDPERMUTATIONSYMMETRIES$_M(M_1, M_2)| - |\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}| + 1$ permutation symmetries are tested using P_M^{Approx}. The experimental results presented later in this section show that for the two CP-nets used in the experiments the approximation is very close (or even equal) to the exact set of permutation symmetries mapping M_1 to M_2. Hence, the number of permutation symmetries which have to be tested is very low in practice. Furthermore, if the multi-sets of coefficients are different for markings no permutation symmetries have to be tested to determine that the markings are not symmetric. It should be noted that a necessary requirement for two markings M_1 and M_2 to be symmetric is that CFMS$(M_1(p_i))$ =CFMS$(M_2(p_i))$ for all places $p_i \in P$ (also for places with structured colour sets). Hence, an obvious way to improve P_M^{Approx} is to test the equality of multi-sets of coefficients for places with structured colour sets before checking any permutation symmetries. Places with atomic colour sets are already accounted for in the approximation.

9.5.2 Experimental Results of the P_M^{Approx} Algorithm

In this section we will present experimental results obtained using an implementation of P_M based on the P_M^{Approx} algorithm. The approximation operated directly on the restriction sets and the approximate set Ψ_{M_1,M_2} is also represented as a restriction set. Before checking the permutation symmetries in Ψ_{M_1,M_2} the approximated set is represented as a list. The permutation symmetries from the list are removed and checked until a permutation symmetry ϕ is found for which $\phi(M_1) = M_2$ or the list is empty. The experimental results are obtained using the two CP-nets presented in Sect. 9.4.

Table 9.2 presents generation statistics for SSPSs for different configurations of the two CP-nets using the P_M^{Approx} algorithm. The first four columns are the same as the first four columns in Table 9.1 presenting the generation statistics using the P_M^{Basic} algorithm. The CP-net column gives the name and configuration of the CP-net for which the SSPS is generated as well as the number of permutation symmetries given by the symmetry specification. The Count column gives two numbers: the total number of times the P_M predicate is called during calculation of the SSPS and the number of calls of P_M which evaluate to true, i.e., the number of calls which determines that the two markings are symmetric.

The last six columns are specific to the P_M^{approx} algorithm. The Cfms column gives the number of calls of P_M^{Approx} for which the multi-sets of coefficients are different for the two markings, i.e., the number of calls of P_M^{Approx} where no permutation symmetries are tested. The Tests column presents statistics on the number of permutation symmetries applied to markings during generation of the SSPS: Total gives the total number of permutation symmetries applied to markings during generation of the SSPS, P_M^{Approx} gives the average number of permutation symmetries applied in each call of P_M^{Approx}, P_M^{Approx}=`true` gives the average number of permutation symmetries applied in each call of P_M^{Approx} which evaluates to true (the case where iteration of the entire Φ_{SG} potentially is avoided), and finally, $\%|\Phi_{SG}|$ gives the average percentage of the permutation symmetries which are tested in a call of P_M^{Approx} during generation of the entire SSPS. Finally, the Time column gives the number of seconds it took to generate the SSPS for the CP-net in the given configuration.

From Table 9.2 it can be seen that checking the multi-sets of coefficients before testing any permutation symmetries in P_M^{Approx} reduces the number of permutation symmetries tested compared to P_M^{Basic}. It is worth noticing that in all calls of P_M^{Approx} which evaluated to false no permutation symmetries are tested, i.e., in all cases the multi-sets of coefficients differ. This is of course highly dependent on the CPN model and is a question of the amount of redundancy encoded in markings of places with structured colour sets. Furthermore, all calls of P_M^{Approx} which evaluated to true only in average requires one permutation symmetry to be tested. This is not a general fact of the technique. However, it is our experience from other experiments that in practice many CP-nets contains a degree of redundancy such that the approximation based on the atomic colour sets of the CP-net often is very close (or equal) to the exact set of permutation symmetries mapping one marking to another.

Even though the number of permutation symmetries tested after approx-

				P_M^{Approx}					
CP-net		Count		cfms	Tests				Time
Con.	$\|\Phi_{SG}\|$	P_M	P_M=`true`		Total	P_M^{Approx}	P_M^{Approx} =`true`	$\%\|\Phi_{SG}\|$	Secs
C_2	2	11	7	4	7	1	1	50.0	0
C_3	6	26	19	7	19	1	1	16.7	0
C_4	24	53	41	12	41	1	1	4.17	0
C_5	120	95	76	19	76	1	1	0.83	0
C_6	720	157	127	30	127	1	1	0.14	1
C_7	5,040	242	197	45	197	1	1	0.02	60
C_8	40,320	–	–	–	–	–	–	–	–
D_2	2	4	2	2	2	1	1	50	0
D_3	6	14	6	8	6	1	1	16.67	0
D_4	24	35	15	20	15	1	1	4.17	0
D_5	120	71	31	40	31	1	1	0.83	0
D_6	720	126	56	70	56	1	1	0.14	0
D_7	5,040	204	92	112	92	1	1	0.02	7
D_8	40,320	–	–	–	–	–	–	–	–

Table 9.2: Generation statistics for SSPS generation using the P_M^{Approx} algorithm.

imating the set of permutation symmetries is 1 for both CP-nets in all configurations it can be seen that SSPSs could not be generated for more than 7 database managers or workers. The reason is the memory required by P_M^{Approx}: in the implementation of P_M^{Approx} used for the practical experiments the approximated set of permutation symmetries is listed before testing the permutation symmetries. Hence, even though the approximation determines the exact set of permutation symmetries in worst case $|\Phi_{SG}|$ permutation symmetries are listed. Thus, in order to make the method applicable in practice we need to carefully consider the representation of the sets of permutation symmetries during generation of the SSPSs. This is the topic of the next section.

9.6 Lazy Listing

In the previous sections we have used sets of restriction sets to represent sets of permutation symmetries. The approximation technique presented in Sect. 9.5 operates directly on the restriction sets. However, in the implementations of both P_M^{Basic} and P_M^{Approx} the permutation symmetries are listed before they are checked. The major drawback of the approach presented in the previous section is that even though the approximation is exact or very close to $\{\phi \in \Phi_{SG} \mid \phi(M_1) = M_2\}$, i.e., only few permutation symmetries have to be checked, the entire approximated set is listed. The experimental results presented in Sect. 9.5 also showed that this approach is not applicable in practice since the memory use becomes a serious bottleneck as system parameters grow. The main goal of this section is therefore to improve the P_M^{Approx} algorithm such that a compact representation of the approximated set of permutation symmetries is maintained during calculation of P_M^{Approx}.

In this section we will present an algorithm which is an improved implementation of the predicate P_M^{Approx}. Section 9.6.1 presents the algorithm. Section 9.6.2 presents experimental results obtained using the Design/CPN OPS tool where the improved algorithm presented in this section is used to determine symmetry between markings.

9.6.1 Presentation of the Algorithm

One way of viewing a set of permutation symmetries (represented as a set of restriction sets) is as a tree. Each level in the tree corresponds to possible images of a given colour. Hence, leafs in the tree represent the permutation symmetries given by the permutation of the individual elements found by following the path from the root to the leaf. Figure 9.3 shows a tree representing Φ_{SG} for the CPN model of the distributed database with 3 database managers and a consistent symmetry specification SG which allow all possible permutations of the atomic colour set `DBM`. The leafs of the tree represent the 6 different permutations symmetries in Φ_{SG}. Below each leaf the permutation symmetry is represented by a restriction set.

When testing a set of permutation symmetries on a marking in the P_M^{Basic} and P_M^{Approx} algorithms we first unfolded the restriction sets to a list of per-

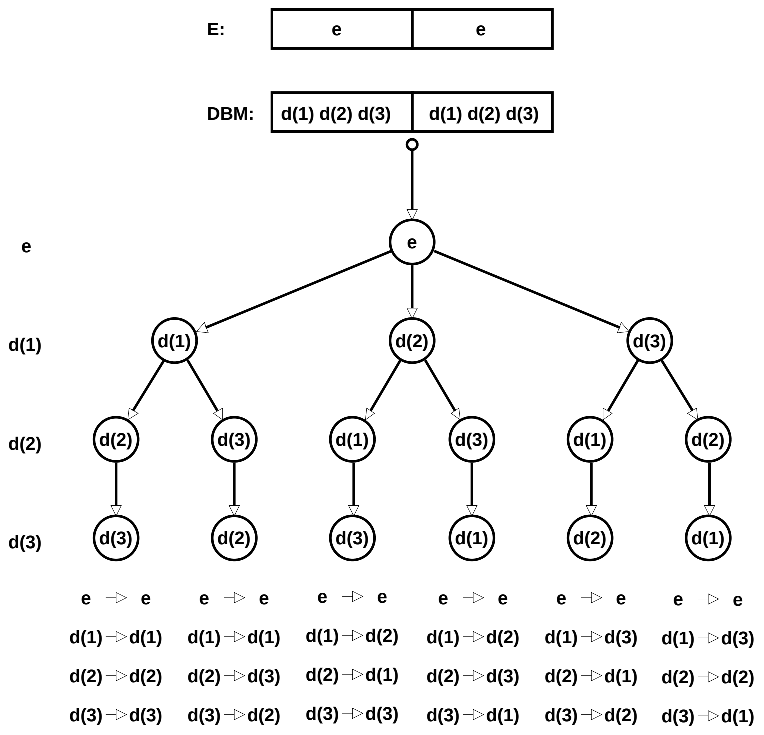


Figure 9.3: All permutation symmetries in Φ_{SG} represented as a tree.

mutation symmetries and then applied the permutation symmetries from an end (until one was found or the entire set was checked). With the approach presented in this section we instead make recursive unfoldings of restriction sets based on a depth first generation of the 'tree view'; each node in the tree corresponds to a recursive call. Each time a leaf is reached the corresponding permutation symmetry is checked. If the permutation symmetry is a symmetry between the two markings checked we conclude that the markings are symmetric (and the iteration stops) otherwise the permutation symmetry is thrown away and the algorithm backtracks to generate the next permutation symmetry. In this way at most one permutation symmetry is contained in memory at a time. In a recursive call corresponding to the ith layer of the tree the algorithm only needs to keep track of the restriction set in the root as well as the images of the colours corresponding to the layers $1,..,(i-1)$. Hence, instead of listing potentially $\prod_{A\in\Sigma_A} |A|!$ permutation symmetries the algorithm needs to represent in the worst case images of at most $\Sigma_{A\in\Sigma_A} |A|$ colours plus the restriction set in the root. An algorithm for such lazy listing of permutations symmetries represented by sets of restriction sets is shown below.

Algorithm: LazyList (i,Φ',M_1,M_2)

```
1: if SinglePermutationSymmetry($\Phi'$) then
2:     $\phi \leftarrow$ GetPermutationSymmetry($\Phi'$)
3:     return TestPermutationSymmetry($\phi, M_1, M_2$)
```

```
4: else
5:    $col \leftarrow$ GetColour($i$)
6:    $images \leftarrow$ GetImages($\Phi'$,$col$)
7:    $found \leftarrow false$
8:    repeat
9:       select $col' \in images$
10:      $images \leftarrow images \setminus \{col'\}$
11:      $found \leftarrow$ LazyList($i+1$,Split($\Phi'$,$col$,$col'$,$M_1$,$M_2$))
12:   until $images = \emptyset \vee found =$ true
13: end if
14: return $found$
```

The algorithm takes four arguments: i is the depth of the call (corresponds to the level in the tree), Φ' is a set of restriction sets representing a set of permutation symmetries, and M_1 and M_2 are the two markings which are checked. First LazyList (i,Φ',M_1,M_2) tests whether the set of restriction sets Φ' given as input represents a single permutation symmetry (line 1). If this is the case a leaf in the tree is reached and the result of applying the permutation symmetry is returned (lines 2-3). If the set of restriction sets represents more than one permutation symmetry (line 4) we have reached an internal node in the tree and a number of depth-first recursive calls are made (lines 8-12). The algorithm uses a number of functions which we will briefly describe below.

SinglePermutationSymmetry(Φ') returns true if Φ' represents a set of a single permutation symmetry and false otherwise.

GetPermutationSymmetry(Φ') returns one of the permutation symmetries in the set represented by the set of restriction sets Φ'.

TestPermutationSymmetry(ϕ, M_1, M_2) tests whether $\phi(M_1) = M_2$.

GetColour(i) returns the colour associated to the i'th level in the tree.

GetImages(Φ',col) returns the possible images of *col*, i.e., the right-hand side of the restriction in Φ in which *col* is contained in the left-hand side.

Split(Φ',col,col') returns a new set of restriction sets which is similar to Φ' except that the restriction containing *col* has been split into two: one containing *col* in the left-hand side and *col'* in the right-hand side and one containing the remaining colours.

An algorithm combining approximation and lazy listing in the symmetry check between markings is given below.

Algorithm: $P_M^{Approx+Lazy}(M_1,M_2)$

```
1: $\Phi' \leftarrow$ FindPermutationSymmetries$_M$ ($M_1$,$M_2$)
2: return LazyList (1,$\Phi'$,$M_1$,$M_2$)
```

CP-net	$\lvert\Phi_S G\rvert$	Time (secs)		
		P_M^{Basic}	P_M^{Approx}	$P_M^{Approx+Lazy}$
C_2	2	0	0	0
C_3	6	0	0	0
C_4	24	0	0	0
C_5	20	0	0	0
C_6	720	23	1	0
C_7	5,040	–	60	1
C_8	40,320	–	–	1
C_9	362,880	–	–	2
C_{10}	3,628,800	–	–	3
C_{15}	$1.3 \cdot 10^{12}$	–	–	77
D_2	2	0	0	0
D_3	6	0	0	0
D_4	24	0	0	0
D_5	120	0	0	0
D_6	720	16	0	0
D_7	5,040	–	7	1
D_8	40,320	–	–	2
D_9	362,880	–	–	4
D_{10}	3,628,800	–	–	8
D_{12}	$4.7 \cdot 10^{8}$	–	–	30
D_{15}	$1.3 \cdot 10^{12}$	–	–	151

Table 9.3: Generation statistics for SSPS generation using the $P_M^{Approx+Lazy}$ algorithm.

The algorithm approximates the set of permutation symmetries using the technique presented in Sect. 9.5 (line 1). To avoid the lengthy listing the permutation symmetries in the approximated set are checked using the LazyList algorithm (line 2).

9.6.2 Experimental Results of the $P_M^{Approx+Lazy}$ Algorithm

In this section we present experimental results obtained using an implementation of P_M based on the $P_M^{Approx+Lazy}$ algorithm. The implementation represents the approximated set of permutation symmetries as a set of restriction sets. During calculation a compact representation is maintained using depth-first recursive unfoldings.

Table 9.3 presents the generation statistics for the generation of the SSPS for different configurations of the two CP-nets using the $P_M^{Approx+Lazy}$ algorithm. The CP-net column gives the name and configuration of the CP-net for which the SSPS is generated as well as the number of permutation symmetries given by the symmetry specification. The next three columns give the time it took to generate the corresponding SSPS using the three algorithms P_M^{Basic}, P_M^{Approx}, and $P_M^{Approx+Lazy}$, respectively.

From Table 9.3 it can be seen that using the $P_M^{Approx+Lazy}$ algorithm it is possible to generate SSPSs for CP-nets with very large symmetry groups. When applying $P_M^{Approx+Lazy}$ for testing the permutation symmetries the same number of permutation symmetries is of course tested as if the permutation symmetries are listed beforehand using the P_M^{Approx} approach. However, the compact representation maintained during calculation saves space since the permutation symmetries are represented by sets of restriction sets. Hence, when combining the idea of lazy listing with the idea of approximations as presented in Sect. 9.5 significant speed up is gained. The reason is that in practice the approximations are often very close to or even equal to the exact set of symmetries between two markings. Thus, the number of permutation symmetries which have to be tested from large permutation groups is usually very small. Furthermore, the memory use of $P_M^{Approx+Lazy}$ caused by the size Φ_{SG} is no longer a bottleneck of the practical applicability of the method.

9.7 Conclusions

We have presented techniques and algorithms to determine whether two markings of CP-nets are symmetric. The algorithms presented are based on general and model independent techniques. Hence, the algorithms can be automatically generated for arbitrary CP-nets. The techniques are implemented in the Design/CPN OPS tool [64] which automatically generates the predicates P_M and P_{BE} needed for the Design/CPN OE/OS Tool [52, 53]. The Design/CPN OPS tool has been used to conduct the experimental results presented in this paper.

The approximation technique that P_M^{Approx} is based on is introduced in [4, 47]. The contribution of this paper is to automate and implement the technique as well as integrate the technique into SSPS generation. The technique is specific to markings of CP-nets and is as such not general for the symmetry method.

The need for compact representations and avoidance of testing the entire group of permutation symmetries is, however, not specific to CP-nets. The algorithms and experimental results presented in this paper are therefore also relevant in other formalisms than CP-nets.

During SSPS generation we store an arbitrary marking from each equivalence class (the first state from the equivalence class encountered during generation of the SSPS). Another strategy is to calculate a canonical representative for each equivalence class. The symmetry check can then be reduced to a simple equivalence check. In [66] we have presented an algorithm for calculation of canonical markings of CP-nets. The algorithm requires the calculation of the minimal marking obtained as a result of applying a set of permutation symmetries. The algorithm for calculation of canonical markings of CP-nets presented in [66] encounters the same problem as the P_M^{Basic} algorithm presented in Sect. 9.4: applying the entire group of permutation symmetries is unfeasible in practice. In [66] we use algebraic techniques to reduce the number of iterations. Even with the use of algebraic techniques the canonicalization of markings experiences problems in practice due to the memory use required when working with large sets of permutation symmetries. The lazy listing approach presented in this paper can directly be used in the problem studied in [66] and it is envisioned that the lazy listing approach can alleviate the bottleneck caused by the memory use in [66].

It is possible to combine the use of algebraic techniques and the techniques presented in this paper to obtain a solution for P_M. However, since the approximation techniques only applies when two markings are compared the approximation techniques presented in this paper cannot be used directly in the algorithm for calculation of canonical markings of CP-nets.

Chapter 10

Modelling Features and Feature Interactions of Nokia Mobile Phones using Coloured Petri Nets

The paper *Modelling Features and Feature Interactions of Nokia Mobile Phones using Coloured Petri Nets* presented in this chapter has been published as a conference paper [67].

[67] L. Lorentsen, A-P. Tuovinen and J. Xu. Modelling Features and Feature Interactions of Nokia Mobile Phones using Coloured Petri Nets. In *Proceedings of the 23th International Conference on Application and Theory of Petri Nets (ICATPN'2002)*, volume 2360 of Lecture Notes in Computer Science, pages 294–313, Springer-Verlag, 2002.

The contents of this chapter is equal to the conference paper [67] except for minor typographical changes.

Modelling of Features and Feature Interactions in Nokia Mobile Phones using Coloured Petri Nets

Louise Lorentsen[*] Antti-Pekka Tuovinen[†] Jianli Xu[†]

Abstract

This paper reports on the main results from an industrial cooperation project[1]. The project is a joint project between Nokia Research Centre and the CPN group at the University of Aarhus. The purpose of the project was to investigate features and feature interactions in development of Nokia mobile phones through construction of a Coloured Petri Nets (CPN) model. The model is extended with domain-specific graphics and Message Sequence Charts to enable mobile phone user interface designers and software developers who are not familiar with Petri Nets to work with the model. The paper presents the CPN model constructed in the project, describes how domain-specific graphics and Message Sequence Charts are used in simulations of the CPN model, and discusses how the project and in particular the construction of the CPN model has influenced the development process of features in Nokia mobile phones.

10.1 Introduction

Modern mobile phones provide an increasingly complex and diverse set of features. Besides basic communication facilities there is a growing demand for facilities for personal information management, data transmission, entertainment, etc. To support flexibility and smooth operation the user interface (UI) of the mobile phone is designed to support the most frequent and important user tasks, hence enabling many features to interplay and be active at the same time. The dependencies or interplay of features are called *feature interactions* and range from simple usage dependencies to more complex combinations of several independent behaviours.

When the project started, feature interactions were not systematically documented at Nokia and often the most complex feature interactions were not fully understood before the features were implemented. In the design and development of features, focus was often on the behaviour and appearance of individual

[*]Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N. DENMARK, E-mail: `louisel@daimi.au.dk`.

[†]Software Technology Laboratory, Nokia Research Centre P.O. Box 407, FIN-00045 Nokia Group, FINLAND, E-mail: {`antti-pekka.tuovinen@nokia.com,jianli.xu@nokia.com`}`@daimi.au.dk`.

[1]The project is funded by Nokia Mobile Phones.

features. The description of feature interactions was integrated in the description of the individual features involved and did not fully cover or treat feature interactions in a systematic way. This could, in the worst case, lead to costly delays in the integration phase of a set of independently developed features. Therefore, a need for more focus on the feature interactions in the development of features was identified.

The above mentioned problems have motivated the MAFIA[2] project. The main goals of the project were:

1. To increase the level of understanding of the role that feature interactions play in the UI software and its development by identification and documentation of typical patterns of feature interactions.

2. To develop a systematic methodology for describing feature interactions.

3. To provide an environment for interactive exploration and simulation of the feature interactions for demonstrational or analytical purposes.

One of the main activities in the MAFIA project was to construct a model using Coloured Petri Nets (CP-nets or CPNs) [46,58]. The use of CP-nets allowed us to have both a static graphical description of features and furthermore allowed simulation of the models and hence provided an environment for interactive exploration and simulation of feature interactions. CP-nets have been used in previous projects within Nokia, e.g., [94], and were therefore known to the project members in Nokia Research Centre.

The paper is structured as follows. Section 10.2 presents the MAFIA project and its organisation. Section 10.3 describes the CPN model constructed in the project. Section 10.4 discusses how the CPN model has been extended with domain-specific graphics and Message Sequence Charts. Section 10.5 contains a discussion of related and future work. Finally, Sect. 10.6 contains the conclusions and a discussion of how the construction of the CPN model has influenced the development of features in Nokia mobile phones. The reader is assumed to be familiar with the basic ideas of High-level Petri Nets [35].

10.2 The MAFIA Project

The MAFIA project was a joint project between Nokia Research Centre and the CPN group at the University of Aarhus. The project ran from November 2000 to November 2001 with a total amount of 15 man months of work resources. The project group consisted of two people from Nokia Research Centre and three people from the CPN group. Hence, the project group consisted of both experts from the application domain, i.e., development of mobile phones, and experts in the tools and techniques to be applied, i.e., CP-nets and its supporting tool Design/CPN [26].

[2]MAFIA is an acronym for Modelling and Analysis of Feature Interaction patterns in mobile phone software Architectures.

Before the joint project started, initial work was done at Nokia Research Centre to experiment with the use of CP-nets for the modelling of feature interactions. The researchers at Nokia Research Centre had practical experience with CP-nets and the Design/CPN tool from other Nokia research projects, e.g., [94]. Hence, the modelling work started immediately in the beginning of the project. When the joint project started, one researcher from the CPN group worked full time at Nokia Research Centre for six months. Other project members from the CPN group provided guidance and technical support on the modelling work.

In the first phase of the project overview information and internal documentation about the features and the mobile phone UI architecture were provided by Nokia and the project group experimented with different levels of abstraction in the CPN model. During the construction of the CPN model, the model was presented and discussed with both UI designers and software developers who were not familiar with Petri Nets. Therefore, already in the first phase of the project the project group decided to experiment with ideas for extending the CPN model with a layer of domain-specific graphics and Message Sequence Charts (MSCs) [45]. This was envisioned to enable the UI designers and software developers to work with the CPN model without interacting with the underlying CP-nets. Interactive simulations were used to validate the models. After the first phase of the project, the CPN modelling proceeded in three iterative steps each containing the following activities.

Refinement and extension. The CPN model was refined and more features were added.

Validation. The CPN model was validated by means of both interactive and more automatic simulations.

Improvement of visualisation facilities. Both the domain-specific graphics and the use of Message Sequence Charts were improved and extended.

Presentation. During meetings with UI designers and software developers the CPN model was presented and its intended use was discussed.

Correction. Based on the evaluation in the meetings with the UI designers and software developers, the CPN model was corrected.

Hence, the CPN modelling proceeded with developing more and more elaborated models, each presented to the UI designers and software developers, i.e., the intended users of the CPN model. At the end of the project the CPN model and documentation were handed over to the users as a part of the results from the project. In Sect. 10.6 we will discuss how the CPN model relates to the other results from the MAFIA project and how the construction of the CPN model has influenced the development of features in Nokia.

10.3 The CPN Model

This section presents parts of the CPN model. The CPN model does not capture the full mobile phone UI software system but concentrates on a number of selected features that are interesting seen from a feature interaction perspective. The purpose of this section is twofold. Firstly, to give an overview of the CPN model and secondly, to give an idea of the complexity and the abstraction level chosen. Section 10.3.1 presents an overview of the CPN model of the mobile phone UI software system. Section 10.3.2 describes how the individual features are modelled and how the similarities in communication patterns can be exploited through reuse of subpages of the CPN model.

10.3.1 Overview of the CPN model

The CPN model has been hierarchically structured into 25 modules (subnets). The subnets are in CPN terminology also referred to as pages, and we will use this terminology throughout the paper.

The page Nokia, depicted in Fig. 10.1, is the top-most page of the CPN model and hence provides the most abstract view of the mobile phone UI software system. UIController, Servers, CommunicationKernel and Applications are all substitution transitions which means that the detailed behaviours are shown on the subpages with the corresponding names.
The four substitution transitions correspond to four concepts of the mobile phone UI software system: *applications*, *servers*, *UI controller*, and *communication kernel.*

Applications. The applications implement the features of the mobile phone, e.g., calls, games and calendar. The terms feature and application will be used interchangeably in the rest of the paper.

Servers. Servers manage the resources of the mobile phone, e.g., the battery, and implement the basic capabilities of the phone.

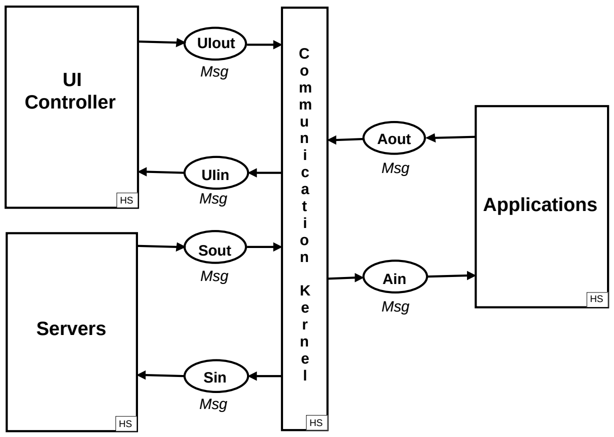


Figure 10.1: Page Nokia.

UI controller. Applications make the feature available to the user via a user interface. The user interfaces are handled by the UI controller. Servers do not have user interfaces. We will sketch the role of the UI controller by means of an example later in this section.

Communication kernel. Servers and applications communicate by means of asynchronous message passing. The messages are sent through the communication kernel which implements the control mechanism and protocol used in the communication between the applications, servers and UI controller.

The focus in the MAFIA project is on features and feature interactions. Hence, we will not go into detail on the modelling of the UI controller, servers and communication kernel. However, we will informally sketch the role of the UI controller as it is necessary for the presentation of the CPN modelling of the applications (features).

The main role of the UI controller is to handle the user interfaces of the applications, i.e., to put the user interfaces of the applications on the display of the mobile phone and to forward key presses from the user to the right applications. Often several applications are active at the same time, for example, when the user is playing a game and a call comes in. We therefore need a priority scheme to determine which application will get access to the display and keys of the mobile phone. This is also the role of the UI controller. In Fig. 10.2 we give a small example to illustrate the operation of the UI controller. The example shows a scenario where the user of the mobile phone starts playing a game. While playing the game a call comes in. The scenario is shown using a Message Sequence Chart (MSC), which is automatically generated during simulation of the CPN model.

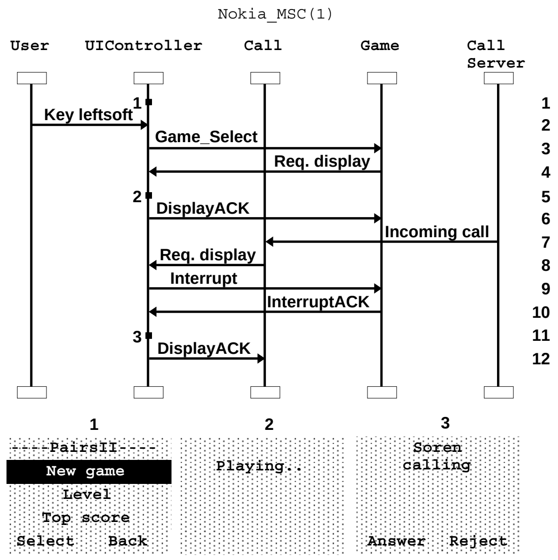


Figure 10.2: Operation of the UI controller.

The MSC contains a vertical line for the user, the UI controller and each of the applications and servers involved in the scenario (here the Game application, the Call application and the Call server). The marks on the vertical line corresponding to the UI controller indicate that the display is updated. A snapshot of the resulting display is shown below the MSC. The number next to the marks and displays indicates the correspondence. Arcs between the vertical lines correspond to messages sent in the mobile phone UI software system. The line numbers in the description below refer to the line numbers in the right-hand side of the MSC.

- The display of the mobile phone when the scenario starts is shown (line 1).

- The user presses the *left soft* key (the key just below the left-hand side of the display) to start a new game (line 2), and the Game application is notified (line 3).

- The Game application requests the display (line 4), the display is updated (line 5), and the Game application is acknowledged (line 6).

- An incoming call arrives (line 7), and the Call application requests the display (line 8).

- The Call application has higher priority than the Game application, hence the game is interrupted (line 9), the Game application acknowledges the interruption (line 10), and the display is granted to the Call application (line 11).

- Finally, the Call application is acknowledged (line 12).

The basic idea behind the model of the UI controller is that the UI controller maintains a structure of the displays (the *display structure*) that the applications in the system have requested to be put on the display of the mobile phone. Due to the limited UI resources of the mobile phone, not all of the display requests can be fulfilled. The displays are assigned a priority to assure that more important features, e.g., an incoming call, will interrupt less important features, e.g., a game. For a given priority the display structure contains a stack of display requests, i.e., if two applications request the display with the same priority the latest request will get the display. Hence, when applications request the UI controller to use the display of the mobile phone, the UI controller will put the display request in the display structure (on top of the stack in the corresponding priority). The UI controller will grant the display to the top of the first, i.e., highest priority, non-empty stack in the display structure. Figure 10.3 shows an example of the display structure where three priorities are shown: foreground, desktop and background with three, four and two display requests in the corresponding stacks, respectively. The display of the mobile phone will indicate that there is an incoming call (the top element of the stack with foreground priority).

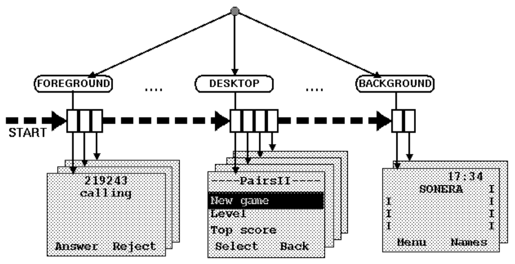


Figure 10.3: Stack like structure of application displays.

10.3.2 Modelling of the Features

We will now go into more detail about the modelling of the features of the mobile phone. In the following we will use the Call feature as an example.

Figure 10.4 shows the detailed behaviour of the page Call modelling the Call feature. The Call feature is an essential feature of the mobile phone and is also the most complex feature included in the CPN model. To enhance readability the Call feature has been divided into a number of subpages. First we will give a brief overview of the Call feature, then we will concentrate on the part of the Call feature handling an incoming call.

The two places, Ain and Aout in the upper left corner in Fig. 10.4, model the message buffers to/from the Call application. It should be noted that there is an arc from Ain to each of the transitions in the page and an arc to Aout from each of the transitions in the page. However, to increase readability these arcs have been hidden in the figure.

The three places Idle, InCall (one active call) and In2Call (two active calls)

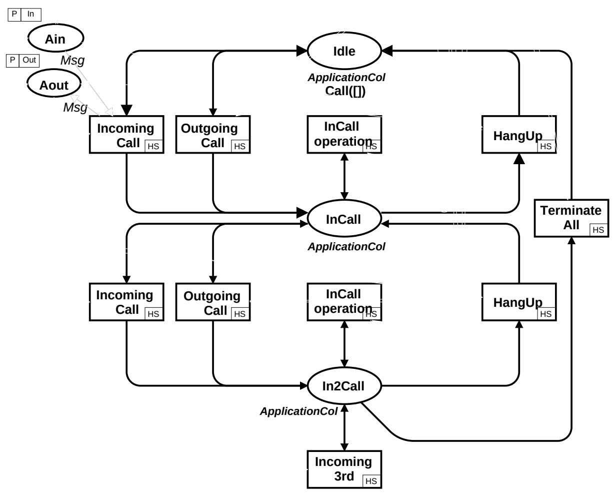


Figure 10.4: Page Call.

model possible states of the Call application. Initially, the Call application is Idle. A new call can be established as either an Incoming Call or an Outgoing Call making the Call application change its state from Idle to InCall. When the Call feature is InCall, a second call can be established as either an Incoming Call or an Outgoing Call making the Call application change its state from InCall to In2Call. In InCall and In2Call a single call can be terminated (modelled by the substitution transitions Hangup) making the Call applications change its state to Idle or InCall, respectively. When the Call application is In2Call, both calls can be terminated (modelled by transition Terminate All) causing the Call application to change its state to Idle. When the Call feature has two active calls, i.e., is in state In2Call a third incoming call can be received (modelled by transition Incoming3rd). The third call cannot be established before one of the two active calls are terminated. Hence, the Call application will remain in In2Call.

Figure 10.5 shows the page IncomingCall which models the part of the Call application handling the reception of an incoming call. This is an example of the most detailed level of the CPN model of the features. As before, there is an arc from Ain to each of the transitions in the page and an arc to Aout from each of the transitions in the page. The arcs from/to the two places Ain and Aout have again been hidden to increase readability of the figure.

The places Idle, Indicating and InCall all have colour set ApplicationCol, which denotes an application and some internal data of the application (here the call and the internal data of the call, i.e., the number of the caller and the status of the call). These three places model the possible states of the Call feature related to the reception of an incoming call. We will explain the rest of the places (the three Init places) later.

In Fig. 10.5 the arrival of an incoming call is modelled by the substitution

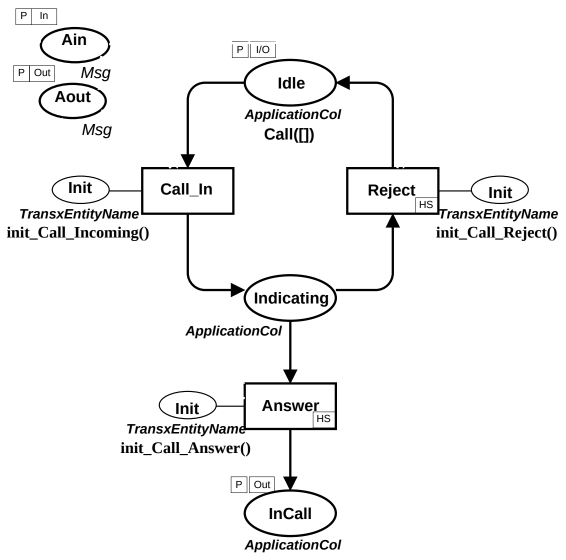


Figure 10.5: Page IncomingCall.

transition Call_In, which causes the Call application to change its state from being Idle to Indicating. Now the call can be either answered or rejected by the user. The substitution transition Reject models the rejection of the incoming call and causes the Call application to change its state back to Idle. The substitution transition Answer models the answering of the call and causes the Call application to change its state from being Indicating to InCall. All the transitions in Fig. 10.5 are substitution transitions, hence the detailed behaviour is modelled on separate pages.

In the following we will give a detailed explanation of the substitution transition Call_In modelling an incoming call to the mobile phone. Figure 10.6 shows in detail what happens when an incoming call arrives in the mobile phone UI software system. The figure is a MSC generated from a simulation of the CPN model. Below we give an textual description. The line numbers refer to the line numbers in the right-hand side of the MSC.

The MSC contains a vertical line for the user, the UI controller and each of the applications and servers involved in the scenario (here the Call application, the Phonebook application, the Any key answer application, the Keyguard application and the Call server). When the simulation starts the phone is idle (line 1).

- The Call application is triggered by an incoming call (via the Call server) (line 2).

- The Call application requests the UI controller, servers, and other applications in the mobile phone UI software system and uses the results to carry out its own task, i.e., to receive and notify the user about the incoming call.

  - The Phonebook application is requested to check whether the calling

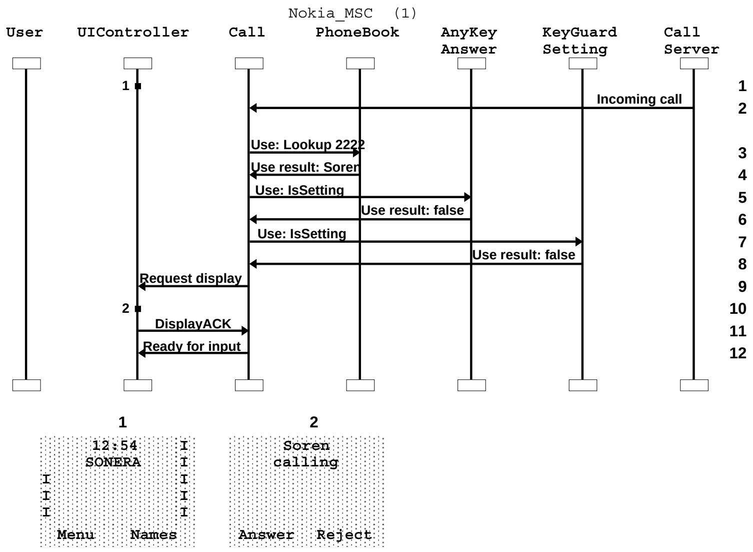


Figure 10.6: An incoming call arrives in the mobile phone.

number is listed in the phonebook (in that case the name of the caller will be written in the display instead of the calling number) (lines 3-4).

– The Any key answer application is requested to check whether the Any key answer setting is on (in that case the call application will allow the call to be answered using any key of the mobile phone, otherwise the call can only be answered using the *offhook* key, i.e., the key with the red phone) (lines 5-6).

– The Keyguard application is requested to check whether it is active (in that case the call can only be answered using the *offhook* key independent of the state of the Any key answer application) (lines 7-8).

– The UI controller is requested to put the user interface of the Call application on the display of the phone and to send information about user key presses to the Call application (line 9). The UI controller updates the display (line 10). Finally, the Call application is acknowledged (line 11). Note that the number was listed in the phonebook (the display indicates that Soren is calling), any key answer was not set, i.e., only information about the *offhook* key should be sent to the Call application (this cannot be seen from the figure).

• The call application changes its state and notifies the UI controller (line 12).

In the above description of the Call application's handling of the incoming call, the events and messages sent in the mobile phone UI software system have been divided into three parts: *trigger*, *request/result loop*, and *proceed*. In fact all state changes of the Call application (as well as the rest of the applications in the mobile phone UI software system) can be characterised by the same three steps. Hence, all of the state changes of the applications are modelled using instances of the same subpage modelling the general communication pattern sketched above. The page modelling the three steps is shown in Fig. 10.7.

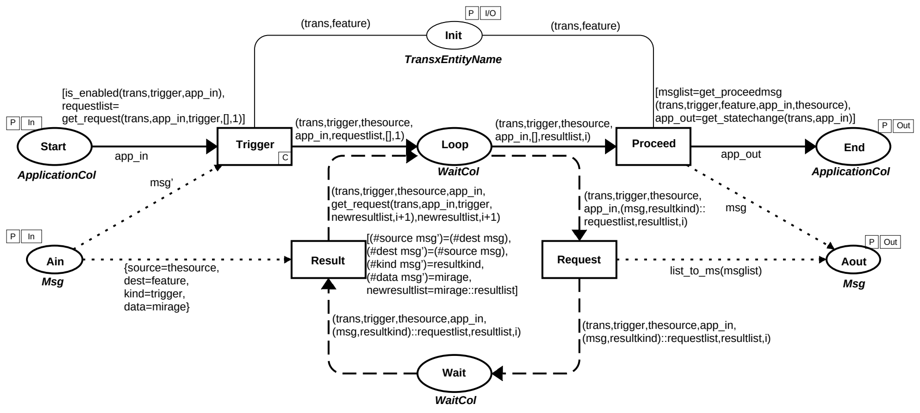

P I/O
(trans,feature)
Init
(trans,feature)
TransxEntityName
[is_enabled(trans,trigger,app_in),
requestlist=
get_request(trans,app_in,trigger,[],1)]
[msglist=get_proceedmsg
(trans,trigger,feature,app_in,thesource),
app_out=get_statechange(trans,app_in)]
P In
Start
app_in
ApplicationCol
Trigger
C
(trans,trigger,thesource,
app_in,requestlist,[],1)
Loop
WaitCol
(trans,trigger,thesource,
app_in,[],resultlist,i)
Proceed
app_out
P Out
End
ApplicationCol
msg'
(trans,trigger,thesource,app_in,
get_request(trans,app_in,trigger,
newresultlist,i+1),newresultlist,i+1)
(trans,trigger,thesource,
app_in,(msg,resultkind)::
requestlist,resultlist,i)
msg
P In
Ain
Msg
{source=thesource,
dest=feature,
kind=trigger,
data=mirage}
Result
[(#source msg')=(#dest msg),
(#dest msg')=(#source msg),
(#kind msg')=resultkind,
(#data msg')=mirage,
newresultlist=mirage::resultlist]
Request
list_to_ms(msglist)
P Out
Aout
Msg
(trans,trigger,thesource,app_in,
(msg,resultkind)::requestlist,resultlist,i)
(trans,trigger,thesource,app_in,
(msg,resultkind)::requestlist,resultlist,i)
Wait
WaitCol

Figure 10.7: Page GenerelSendReceive.

The page GenerelSendReceive contains five *port places*: Start, End, Ain, Aout and Init. The two places Start and End (in the left and right of Fig. 10.7) are bound to the input place and the output place of the substitution transition (when the page is used for the Call_In substitution transition in Fig. 10.5, Start and End are bound to Idle and Indicating). Places Ain and Aout are bound to the message buffers to/from the application. Place Init is bound to the place connected to the substitution transition with a thin line (a line corresponds to an arc with arrowheads in both directions).

- The first step (trigger) is modelled by the Trigger transition modelling that the state change of the application is triggered by some event sent to the application via the Ain place causing the application to change its state from Start to Loop.
- The second step (request/result loop) is modelled by the two transitions Request and Result (the loop in Fig. 10.7 with the dashed arcs). Request models the sending of a request to another entity in the system (an application, server or the UI controller) via the Aout buffer and causes the application to change its state from Loop to Wait. When a result appears on place Ain transition Result can occur, causing the application to change its state back to Loop. The loop can be executed several times until the application has no more requests.
- The third step (proceed) is modelled by the transition Proceed modelling that a message is sent to the Aout buffer. The application changes its state to End, thus modelling a complete state change of the application.

The concrete information about which messages trigger the event, which requests are sent, how the results are used by the application, and what change in data are performed, etc. is determined from the (initial) marking of the Init place. The same page (shown in Fig. 10.7) is used as a subpage for all of the substitution transitions modelling state changes of the applications with individual installations of the corresponding Init places. The reuse of the page shown in Fig. 10.7 means that even though the CPN model of the mobile phone UI software system only consists of 25 different pages it actually contains 98 page instances.

We have now presented the CPN model of the Call application to give an idea about the complexity of the CPN model and the efforts required to include a feature in the model. All features in the CPN model follow the same idea as the Call feature. Hence, when adding a new feature to the CPN model, a page modelling the overall state changes is provided together with a description of triggers, requests/results and proceed messages. This way of modelling the features reflects the way the features are currently documented in the written specifications of features in Nokia mobile phones. Here the features are described using a textual description of the possible states and state changes of the feature together with a description of the user interfaces and the use of other features and the reactions of the feature to user key presses. Hence, the CPN model closely follows both the UI designers' and software developers' current understanding of features as well as the current available documentation of the features.

10.4 Simulations and Visualisation

We model the individual features of the mobile phone using the same ideas as described for the Call feature. The feature interactions are captured in the CPN model as the communication and interaction between the individual features in the CPN model. The UI designers and software developers who are developing new features will use the CPN model to identify and analyse the interaction patterns of their new features as well as existing features. One way of using the CPN model is by means of simulations; both interactively (step-by-step) and more automatically for investigations of larger scenarios. In this section we will present techniques which allow UI designers and software developers who are not familiar with Petri Nets to control and gain information from simulations without directly interacting with the underlying CP-net and its token game.

Two extensions have been made to the CPN model allowing the current state and behaviour of the CPN model to be visualised during simulations. Firstly, the state of the phone as the user observes it on the handset is visualised via an animation of the display. Secondly, the CPN model is extended with Message Sequence Charts (MSCs) which are automatically constructed as graphical feedback from simulations. MSCs were chosen to visualise the behaviour of the CPN model because diagrams similar to MSCs are already in use in the design process at Nokia. Therefore, MSCs allow the behaviour of the CPN model to be visualised in a way that is familiar to the UI designers and software developers. The use of domain-specific graphics and MSCs allow the state of the model to be visualised. However, the users of the CPN model are also interested in controlling the simulations without interacting directly with the underlying CP-nets, i.e., without explicitly selecting the transitions and bindings to occur. The CPN model has therefore also been extended with facilities for controlling the simulations without interacting with the underlying CP-nets.

In the following we will present techniques for visualising information about the simulation and for controlling the simulation using domain-specific graphics. Figure 10.8 shows how the domain-specific graphics appears as a single page in the CPN model of the mobile phone UI software system. In the following we will present the elements of the page shown in Fig. 10.8 in detail.

10.4.1 Animation of the display

The CPN model is extended with a picture of a mobile phone. The picture is used both to present information about the simulation to the user and to get information from the user to control and guide simulations of the CPN model. The state of the phone as the user observes it via the handset is visualised via an animation of the display. The left-hand side of Fig. 10.8 shows a snapshot of the animation taken during a simulation of the CPN model. The snapshot shown corresponds to a state of the CPN model where the Call feature is Indicating.

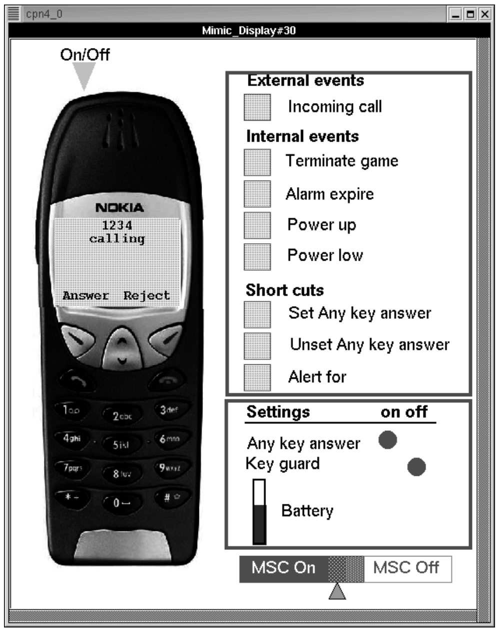


Figure 10.8: The domain-specific graphics for the mobile phone UI software system.

The user now has the possibility to either answer (transition Answer in Fig. 10.5) or reject (transition Reject in Fig. 10.5) the incoming call. The picture of the mobile phone is also used to control the simulation. By means of key presses (using mouse clicks) on the picture of the mobile phone, the user can control the simulation, i.e., start a game or answering an incoming call. In the following we will sketch how the use of domain-specific graphics has been integrated in the CPN model of the mobile phone UI software system.

The use of domain-specific graphics is implemented using the Mimic library [78] of Design/CPN, which allows graphical objects to be displayed, updated and selected during simulations. The animation in the left-hand side of Fig. 10.8 is constructed as number of layered objects:

- A background picture of a Nokia mobile phone is used to make the use of graphics look like a real Nokia mobile phone as it is known to the users of the CPN model.

- On top of the display of the background picture there are a number of regions (one for each area of the mobile phone display). Updating the contents of those regions allow us to animate the display of the mobile phone during simulation. We have implemented a function `update_display()` that scans through the display structure and finds the top element (display request) in the first non-empty stack, i.e., the one to be put on the display of the mobile phone. The regions on top of the display of the background picture are updated with the contents of this "top most" display.

- On top of each of the keys of the background picture there is a region.

We have implemented a function `get_input()` which enables the user to select one of the regions corresponding to the keys of the mobile phone and return information about which region, i.e., which key, is pressed.

The two functions `update_display()` and `get_input()` are called from code segments in the CPN model. Hence, the graphics is updated during simulations; user key presses are also read into the CPN model and can be used to control the simulations.

The animation of the display shows the state of the CPN model as the user of the handset observes it. Often the user is also interested in getting information about the internal state of the CPN model, e.g., the current personal settings of the mobile phone. Hence, the CPN model is also extended with graphics visualising the internal state of the phone. The lower part in the right-hand side of Fig. 10.8 shows how the internal state of the CPN model of the mobile phone UI software system is visualised using the Mimic library. The graphics visualise the state of Any key answer and Keyguard applications (whether they are *on* or *off*) and the amount of power in the battery. The graphics is updated dynamically during simulations to reflect the current state of the CPN model.

10.4.2 Controlling the simulations

We have already described how the user can control the simulations by means of mouse clicks on the picture of the mobile phone. However, this often turns out to be insufficient. There are a number of events which are not generated as a result of user key presses, e.g., an incoming call to the mobile phone. The top-most part in the right-hand side of Fig. 10.8 shows how the CPN model has been extended with the possibility of generating such events by pressing buttons during simulations. The events are divided into three categories:

External events i.e., events that are caused by the environment of the mobile phone (here incoming calls from the network).

Internal events i.e., events that are generated by the mobile phone software without the influence of the user (here termination of a game, expiration of the alarm clock, battery level low warning, and battery charging).

Short cuts allow the user of the CPN model to perform an operation without pressing a series of keys (here set or unset the Any key answer with a single key press instead of through the menu and change which phone numbers to alert for).

10.4.3 Message Sequence Charts

The animation of the display and the visualisation of the internal state of the mobile phone will provide information about the state of the CPN model as the user of the mobile phone will observe it. However, the software developers are also often interested in gaining information about the UI software system at a more detailed level. This is obtained through the use of MSCs which capture the communication between applications, servers and the UI controller in the mobile phone UI software architecture. Figure 10.2 shows an example of a MSC automatically generated from a simulation of the CPN model. The MSC is described in detail in Sect. 10.3.

The MSC shown in Fig. 10.2 has a vertical line for both the user, the UI controller and each of the applications and servers involved in the scenario. During meetings with both UI designers and software developers we identified that the desired level of detail in the MSCs depends highly on the background of the user. Hence, we found it necessary to be able to customise the MSCs to the individual users of the CPN model. Hence, we now provide four possible levels of detail (see the lowest part in the right-hand side of Fig. 10.8). It should be noted that the MSCs can dynamically be turned on and off during simulations. Also the level of detail in the MSC can be changed dynamically. A small triangle below the buttons indicate which level is currently in use.

1. MSC On will generate detailed MSCs with vertical lines corresponding to the user, the UI controller as well as for all applications and servers in the system and arcs for all messages sent in the system.

2. The two buttons in the middle (between MSC On and MSC Off) generate MSCs in two intermediate levels of detail

   - The leftmost button generates MSCs like 1. but omits arcs corresponding to acknowledgement messages. The MSC in Fig. 10.2 is generated using this level of detail.
   - The rightmost button generates MSCs where only the communication between the user and the user interface is captured. This kind of MSC is mostly aimed at the UI designers who design the layout and the use of the display.

3. MSC Off turns off the use of MSCs.

In this section we have presented separate techniques for visualising information about the simulation and for controlling the simulation without interacting directly with the underlying CP-nets. It should be noted that during simulation of the CPN model, the page in Fig. 10.8 is the only page of the CPN model that needs to be visible to the user. Hence, the CPN model can be demonstrated and used without showing the underlying CPN model.

10.5 Related and Future Work

A number of published papers, e.g., [18, 65, 94], report on projects where CP-nets successfully have been used in industrial settings. Feature interactions have been studied in several application areas, e.g., telecommunications systems (see [56] for a survey), process planning [43], and computer-aided design [75]. To our knowledge there are no applications of CP-nets to feature interactions.

Especially in the area of telecommunication services much research work have been done on feature interactions and there is a biannual workshop on the topic [9, 57]. Our work in the MAFIA project does not relate to what is traditionally known as *telecom feature interactions*; our problem is more general: how to coordinate concurrent interrelated services. However, we can draw some parallels. Much of the work in the area of feature interactions in telecommunications, e.g., [3,10,73], concentrate on the use of formal methods to automatically detect feature interactions. This is, however, not the aim of work the MAFIA project. In our work we focus on the development process of features, i.e, how the use of CP-nets to model the basic call model and the additional features can increase the level of understanding of the feature interactions and aid the future development of new features. Thus, we will categorise our approach to belong to the software engineering trend identified in [8] as one of the three major trends in the field of telecommunications services. However, the use of an underlying formal method, i.e., CP-nets, enables us to both obtain a formal description of the features and feature interactions and an environment for interactive exploration and simulation of the feature interactions. Furthermore, the construction of the CPN model as well as simulations were used to resolve inconsistencies in the specifications and to gain knowledge about features and feature interactions.

Based on the above, an interesting question is therefore whether the modelling approach of features developed in the MAFIA project can be used to automatically analyse and detect feature interactions. Using the Design/CPN Occurrence Graph Tool (OG Tool) [17] initial attempts have been done to evaluate the applicability of the CPN model for analysis purposes. State spaces have been generated for different combinations of features (including the features in isolation, i.e., the basic CPN model with only the analysed feature included). Not all combinations of features have been analysed and no attempts have been done to generate the state space for the full CPN model of the mobile phone UI software system presented in Sect. 10.3; only state spaces for selected combinations were generated with the goal of evaluating if (and how) state space analysis can be used to detect feature interactions in the CPN model developed in the MAFIA project.

In the following $\mathcal{S}_{f_1,f_2}$ denotes the full state space of the basic CPN model including the features f_1 and f_2. $\mathcal{S}_f \models P$ means that the property P can be verified from $\mathcal{S}_f$. One possible way of formally expressing that two features f_1 and f_2 interact is that for a property P we have that $\mathcal{S}_{f_1} \models P$ but $\mathcal{S}_{f_1,f_2} \not\models P$. We will use two properties P_1 and P_2 as examples:

P_1= There are no *deadlocks* and the initial state is a *home state*.

P_2= If a Call comes in and the Any Key Answer is set, then pressing any key on the mobile phone will answer the Call.

P_1 and P_2 can be formulated as *queries* in the OG tool. The answers to the queries can then be automatically determined by the OG tool when a state space has been generated. P_1 is general property of the CPN model and can be expressed as a standard query of a CPN model. P_2 relates to specific features of the CPN model and can be formulated using temporal logic [22]. Properties expressed in temporal logic can be verified using the OG tool library ASK-CTL [20] which makes it possible to make queries formulated in a state and action oriented variant of CTL [12]. We will not go into detail with how the properties are expressed as queries. Instead we will show how P_1 and P_2 can be used to detect feature interactions in the CPN model of the mobile phone UI software system. Analysing different combinations of features we find that

1. $\mathcal{S}_{\mathsf{Call}} \models \neg P_1$ but $\mathcal{S}_{\mathsf{Call,AnyKeyAnswer,PhoneBook,KeyGuard}} \not\models \neg P_1$
2. $\mathcal{S}_{\mathsf{AnyKeyAnswer}} \models P_2$ but $\mathcal{S}_{\mathsf{AnyKeyAnswer,KeyGuard}} \not\models P_2$

The first feature interaction found is an interaction between the Call, Any key answer, Phonebook and Keyguard features. The interaction is a result of the Call application's use of other features in the mobile phone software system. Figure 10.6 shows how the Call applications requests the Phonebook application, the Any key answer application and the Keyguard application when an incoming call arrives at the mobile phone. Hence, a deadlock appears when the modelled Call application exists in isolation, i.e., without the Phonebook application, the Any key answer application and the Keyguard application.

The second feature interaction found is an interaction between the Any key answer and Keyguard features. The interaction appears as a result of the features' capability of modifying the behaviour of other features. Here the Keyguard application disables the Any key answer application to prevent the user from answering an incoming call by accident.

After having performed the basic analysis presented in this section we conclude that the CPN model developed in the MAFIA project seems applicable for analysis and detection of feature interactions; no major changes or adjustments are needed. However, we expect the state space of the full CPN model to be very large. Since the features of the mobile phone are asynchronous a possible direction for future analysis would be to use the stubborn set method [88] to reduce the size of the state space.

10.6 Conclusions

This paper describes the MAFIA project and one of its main activities: the construction of a CPN model of the mobile phone UI software system. Previous sections have reported on the CPN model and its construction. In this section we will conclude the paper by discussing how the CPN model was used within the MAFIA project and how the construction of the CPN model has influenced the development process of features in Nokia mobile phones.

The CPN model has been constructed in a period of six months and has been constructed in several iterative steps with more and more elaborated models each presented to the intended users of the CPN model, i.e., the UI designers and software developers. The fact that the CPN model was presented to users not familiar with Petri Nets meant that the CPN model was extended with domain-specific graphics at a very early stage. The graphics was developed and extended in parallel with the underlying CP-net.

An important aspect of the CPN model developed in the MAFIA project is that the model is intended to be used in very different settings:

High level (UI oriented). UI designers design the features and their interactions at the level of the user interface of the mobile phone. The CPN model provides possibilities for simulations with feedback by means of the animation of the display and MSCs with detailed information about the flow of user interaction and snapshots of the contents of the mobile phone display. The UI designers can simulate the CPN model using only the page with Mimic graphics.

Conceptual level. The architecture and protocol for the mobile phone UI software system is designed particularly to be well suited for development of features. The CPN model provides possibilities for simulations with more detailed feedback about the individual features and their interactions (messages sent between the individual applications, servers and the UI controller).

Low level. Software developers specify and implement the new features of mobile phones. The CPN model is developed to make it possible to easily add new features in the model without changing the existing model. Hence, new features can be included in the CPN model and simulated with the features already included in the CPN model.

The construction of the CPN model of the mobile phone UI software system is only one of the activities in the MAFIA project. Other activities include development of a categorisation of feature interactions. During development of such a categorisation, the CPN model was used to experiment with the categories found. The CPN model have also been used to produce static figures (like the MSC in Fig. 10.2) to a document explaining feature interaction to UI designers and UI testers in Nokia. We have already described how the CPN model was constructed in close cooperation with UI designers and software developers of Nokia mobile phones. The UI designers and software developers also suggested features to be included in the CPN model to ensure that the categorisation of feature interactions developed in the MAFIA project is covered by features included in the CPN model. Simulations were used to understand the feature interactions and as a starting point for discussion.

The CPN model is intended to be used in future feature development in Nokia; both as a means for presenting and experimenting with the different features and feature interactions captured in the CPN model, but also as a framework for development of new features. New features can be designed and included in the CPN model and the behaviour and interactions with other features can be observed using simulations of the CPN model. However, even if the CPN model was not to be used in feature development, we find that the project group and the users involved in the development process have benefitted from the construction of the CPN model. During construction of the CPN model, a number of inconsistencies in the descriptions of the features have been identified. The modelling process has identified what kind of information about a feature's interaction with other features need to be provided by the designer of the feature and what kind of problems need to be resolved when a feature is designed. In parallel with the MAFIA project the UI designers involved in the construction of the CPN model were designing a new feature for a Nokia product. During development of the feature, focus was pointed at interactions with other features based on experiences gained from the MAFIA project, and this resulted in a suggestion for new design templates and development practises.

The construction of the CPN model has also generated new ideas for the mobile phone UI architecture. The first version of the CPN model was constructed by the project group based on written documentation. A number of inconsistencies were found, and some of these could not be resolved before the first meeting with the users. Hence, a solution was chosen and presented to the users as a starting point for discussion. In fact, the chosen solution was not the correct solution with respect to the current implementation of the mobile phone UI software system. However, the solution modelled was found to be interesting and in fact a possible suggestion for a change in the mobile phone UI architecture to make it more oriented towards feature interactions.

In conclusion we can say that the MAFIA project has improved knowledge about features and feature interactions and has influenced an ongoing change of design practises of features in new products of Nokia mobile phones. The CPN model developed in the MAFIA project is intended to be used in future design of features of Nokia mobile phones. Furthermore, the modelling activities have raised interesting questions and ideas that can lead to design changes of the mobile phone UI architecture.

Bibliography

[1] R. B. Allenby. *Rings, Fields, and Groups: An Introduction to Abstract Algebra.* Oxford, England: Oxford University Press, 1991.

[2] L. Allison. Generating coset representatives for permutation groups. *Journal of Algorithms*, 1981.

[3] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware. Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems*, volume VI, Amsterdam, May 2000. IOS Press.

[4] R. Andersen, J. Jørgensen, and M. Pedersen. Occurrence Graphs with Equivalent Markings and Self-Symmetries. Master's thesis, Department of Computer Science, University of Aarhus, Denmark, 1991. Only available in Danish: Tilstandsgrafer med ækvivalente mærkninger og selvsymmetrier.

[5] M. Awad, J. Kuusela, and J. Ziegler. *Object-Oriented Technology for Real-Time Systems: A Practical Approach using OMT and Fusion.* Prentice-Hall, 1996.

[6] D. Bosnacki, D. Dams, and L. Holenderski. Symmetric Spin. In *Proceedings of the 7th SPIN Workshop*, volume 1885 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2000.

[7] G. Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *Lecure Notes in Computer Science.* Springer-Verlag, 1991.

[8] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature Interaction: A Critical Review and Considered Forecast. Submitted for publication. On-line version: `http://www.dcs.gla.ac.uk/~muffy/papers/calder-kolberg-magill-reiff.pdf`.

[9] M. Calder and E. Magill. Feature Interactions in Telecommunications and Software Systems VI. IOS Press, 2000.

[10] M. Calder and A. Miller. Using SPIN for Feature Interaction Analysis - a Case Study. In *Proceedings of SPIN 2001*, volume 2057 of *Lecture Notes in Computer Science*, pages 143–162. Springer-Verlag, 2001.

[11] C. Capellmann, S. Christensen, and U. Herzog. Visualising the behaviour of intelligent networks. In T. Margaria, B. Steffen, R. Rückert, and J. Posegga, editors, *Services and Visualization, Towards User-Friendly Design*, volume 1385 of *Lecture Notes in Computer Science*, pages 174–189. Springer-Verlag, 1998.

[12] A. Cheng, S. Christensen, and K. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M. Spathopoulos, R. Smedinga, and P. Kozák, editors, *Proceedings of WODES'96*. Institution of Electrical Engineers, Computing and Control Division, Edinburgh, UK, 1996.

[13] A. Cheng, S. Christensen, and K. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M. Spathopoulos, R. Smedinga, and P. Kozák, editors, *Proceedings WODES '96*. Institution of Electrical Engineers, Computing and Control Division, Edinburgh, UK, 1996.

[14] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and Their Symbolic Reachability Graph. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 373–396. Springer-Verlag, 1991.

[15] J. B. Christensen and L. Kristensen. Verification of Coloured Petri Nets Using State Spaces with Equivalence classes. In W. Aalst, J. Colom, F. Kordon, G. Kotsis, and D. Moldt, editors, *Petri Net Approaches for Modelling and Validation*, chapter 2, pages 17–34. Lincoln Europa, 1999.

[16] S. Christensen. *Message Sequence Charts. User's Manual*, January 1997. Available from http://www.daimi.au.dk/designCPN/.

[17] S. Christensen, K. Jensen, and L. Kristensen. *Design/CPN Occurrence Graph Manual*. Department of Computer Science, University of Aarhus, Denmark.
On-line version: `http://www.daimi.au.dk/designCPN/`.

[18] S. Christensen and J. Jørgensen. Analysis of Bang and Olufsen's BeoLink Audio/Video System Using Coloured Petri Nets. In P. Azéma and G. Balbo, editors, *Proceedings of ICATPN'97*, volume 1248 of *Lecture Notes in Computer Science*, pages 387–406. Springer-Verlag, 1997.

[19] S. Christensen, J. B. Jørgensen, and L. M. Kristensen. Design/CPN - A Computer Tool for Coloured Petri Nets. In E. Brinksma, editor, *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 1997.

[20] S. Christensen and K.H.Mortensen. *Design/CPN ASK-CTL Manual*. Department of Computer Science, University of Aarhus, Denmark, 1996.
Online: `http://www.daimi.au.dk/designCPN/`.

[21] E. Clake, E. Emerson, S. Jha, and A. Sistla. Symmetry Reductions in Model Checking. In A. Hu and M. Vardi, editors, *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 147–159. Springer-Verlag, 1998.

[22] E. Clarke, E. Emerson, and A. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[23] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Logic Model Checking. *Formal Methods in System Design*, 9, 1996.

[24] E. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. In Springer-Verlag, editor, *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science (LNCS)*, pages 450–462. Springer-Verlag, 1993.

[25] J. Day and H. Zimmermann. The OSI Reference Model. *Proceedings of the IEEE*, 71, December 1983.

[26] Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.

[27] E. Emerson. *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 995–1072. Elsevier, 1990.

[28] E. Emerson and A. P. Sistla. Symmetry and Model Checking. *Formal Methods in System Design*, 9, 1996.

[29] E. A. Emerson, editor. *Formal Methods in System Design*, volume 9. Kluwer Academic Publishers, 1996.

[30] E. A. Emerson and A. P. Sistla. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach. In *Proceedings of CAV'95*, volume 939 of *Lecture Notes in Computer Science*, pages 309–324. Springer-Verlag, 1995.

[31] D. Floreani, J. Billington, and A. Dadej. Designing and Verifying a Communications Gateway Using Coloured Petri Nets and Design/CPN. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 153–171. Springer-Verlag, 1996.

[32] A. Foroughipour. Construction of the OS-graph with Permutation Symmetries of a Coloured Petri Net using Algebraic Algorithms. Master's thesis, Master Thesis, Department of Computer Science, Aarhus University, 1994.

[33] G. Gallasch and L. Kristensen. Comms/cpn: A communitation infrastructure for external communication with design/cpn. In K. Jensen, editor, *Proceedings of the Third Workshop on Practical Use of Coloured Petri Nets and the CPN Tools*, DAIMI-PB – 554, pages 75–90, 2001.

[34] The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 1999. (`http://www-gap.dcs.st-and.ac.uk/~gap`).

[35] H. Genrich. Predicate/Transition Nets. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 3–43. Springer-Verlag, 1991.

[36] H. Genrich and R. Shapiro. Formal verification of an arbeiter cascade. In *Proceedings of the 13th International Conference on Application and Theory of Petri Nets*, volume 616 of *Lecture Notes in Computer Science (LNCS)*, pages 205–223. Springer-Verlag, 1992.

[37] J. C. Grégoire. State Space Compression in SPIN with GETSs. In *Proceedings of SPIN'96 Workshop*, pages 90–109, 1996.

[38] V. Gyuris and A. P. Sistla. On-the-Fly Model Checking Under Fairness That Exploits Symmetry. In *Proceeding of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, 1997.

[39] G. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[40] G. J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer*, 2:270–278, 1999.

[41] P. Huber, A. Jensen, L. Jepsen, and K. Jensen. Towards reachability trees for high-level petri nets. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 188 of *Lecture Notes in Computer Science (LNCS)*, pages 215–233. Springer-Verlag, 1984.

[42] P. Huber, A. Jensen, L. Jepsen, and K. Jensen. Reachability trees for high-level petri nets. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets; Theory and Application*, pages 319–350. Springer-Verlag, 1991.

[43] J.-S. Hwang and W. A. Miller. Hybrid Blackboard Model for Feature Interactions in Process Planning. *Computers and Industrial Engineering*, 29(1–4):613–617, 1995.

[44] C. Ip and D. Dill. Better Verification Through Symmetry. *Formal Methods in System Design*, 9, 1996.

[45] ITU (CCITT). Recommendation Z.120: MSC. Technical report, International Telecommunication Union, 1992.

[46] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.

[47] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.

[48] K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9, 1996.

[49] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use.* Monographs in Theoretical Computer Science. Springer-Verlag, 1997. ISBN: 3-540-62867-3.

[50] K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN Reference Manual.* Department of Computer Science, University of Aarhus, Denmark, 1995.
Online: `http://www.daimi.au.dk/designCPN/`.

[51] J. Jørgensen. Construction of Occurrence Graphs with Permutation Symmetries Aided by the Backtrack Method. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1997. DAIMI PB-516, ISSN 0105-8517, February 1997.

[52] J. Jørgensen and L. Kristensen. *Design/CPN OE/OS Graph Manual.* Department of Computer Science, University of Aarhus, Denmark, 1996.
Online: `http://www.daimi.au.dk/designCPN/`.

[53] J. Jørgensen and L. Kristensen. Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):714–732, July 1999.

[54] J. Jørgensen and K. Mortensen. Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[55] T. Junttila. Computational Complexity of the Place/Transition-Net Symmetry Reduction Method. Research report, Helsinki University of Technology, Laboratory for Theoretical Computer Science, apr. 2000.

[56] D. O. Keck and P. J. Kuehn. The Feature and Service Interaction Problem in Telecommunication Systems: A survey. *IEEE Transactions on Software Engineering*, 24(10):779–796, October 1998.

[57] K. Kimbler and L. G. Bouma. Feature Interactions in Telecommunications and Software Systems V. IOS Press, 1998.

[58] L. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, December 1998.

[59] L. M. Kristensen. *State Space Methods.* PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2000.

[60] L. M. Kristensen and A. Valmari. Finding stubborn sets of coloured petri nets without unfolding. In J. Desel and M. Silva, editors, *Proceedings of*

the 19th International Conference on Application and Theory of Petri Nets (ICATPN'98), volume 1420 of *Lecture Notes in Computer Science*, pages 104–123. Springer-Verlag, 1998.

[61] L. M. Kristensen and A. Valmari. Improved question-guided stubborn set methods for state properties. In M. Nielsen and D. Simpson, editors, *Proceedings of the 21th International Conference on Application and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *Lecture Notes in Computer Science*, pages 282–302. Springer-Verlag, 2000.

[62] W. Lawrenz. *CAN Contoller Area Network, Grundlagen und Praxis.* Hüttig Buch Verlag, Heidelberg, 1994.

[63] L. Lorentsen. Coloured petri nets and state space generation with the symmetry method. In K. Jensen, editor, *To appear in Proeedings of the 4th Workshop on Applications og Coloured Petri Nets and the CPN Tools*, 2002.

[64] L. Lorentsen. *Design/CPN OPS Graph Manual.* Department of Computer Science, University of Aarhus, Denmark, 2002.
Online: `http://www.daimi.au.dk/∼louisel/`.

[65] L. Lorentsen and L. Kristensen. Modelling and Analysis of a Danfoss Flowmeter System. In M.Nielsen and D.Simpson, editors, *Proceedings of the 21th International Conference on Application and Theory of Petri Nets (ICATPN'2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 346–366. Springer-Verlag, 2000.

[66] L. Lorentsen and L. Kristensen. Exploiting stabilizers and paralellism in state space generation with the symmetry method. In *Proceedings of the Second International Conference on Application of Concurrency to System Design (ICACSD'01)*, pages 211–220. IEEE, 2001.

[67] L. Lorentsen, A.-P. Tuovinen, and J. Xu. Modelling of Features and Feature Interaction Patterns in Nokia Mobile Phones using Coloured Petri Nets. In J. Esparza and C. Lakos, editors, *Proceedings of the 23rd International Conferece on Theory and Application of Petri Nets 2002 (ICATPN'02)*, volume 2360 of *Lecture Notes in Computer Science (LNCS)*, pages 294–313. Springer-Verlag, 2002.

[68] O. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language.* ACM Press, 1993.

[69] K. McMillan. The SMV System. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1992.

[70] R. Milner, R. Harper, and M. Tofte. *The Definition of Standard ML.* MIT Press, 1990.

[71] M. Mäkelä. Efficiently verifying safety properties with idle office computers. In C. Lakos, R. Esser, L. Kristensen, and J. Billington, editors, *Proceedings of the Workshops on Software Engineering and Formal Methods and Formal Methods Applied to Defence Systems*, volume 12 of *Conferences in Research and Practice in Information Technology*, pages 11–16. Australian Computer Society, 2002.

[72] T. Murata. Petri Nets: Properties, Analysis and Application. In *Proceedings of the IEEE, Vol. 77, No. 4*. IEEE Computer Society, 1989.

[73] M. Nakamura, Y. Kakuda, and T. Kikuno. Feature Interaction Detection using Permutation Symmety. In K. Kimbler and L. G. Bouma, editors, *Feature Interactions in Telecommunications and Software Systems*, volume V, pages 187–201, Amsterdam, September 1998. IOS Press.

[74] The PEP Tool. `http://teoretica.informatik.uni-oldenburg.de/~pep/`.

[75] D.-B. Perng and C.-F. Chang. Resolving Feature Interactions in 3rd Part Editing. *Computer-Aided Design*, 29(10):687–699, 1997.

[76] The PROD Tool. `http://www.tcs.hut.fi/prod/`.

[77] J. Rasmussen and M. Singh. Designing a security system by means of coloured petri nets. In J. Billington and W. Reisig, editors, *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 1996.

[78] J. L. Rasmussen and M. Singh. *Mimic/CPN. A Graphical Simulation Utility for Design/CPN. User's Manual.* On-line version: `http://www.daimi.au.dk/designCPN/`.

[79] W. Reisig. *Petri Nets*, volume 4 of *EACTS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[80] K. Schmidt. How to Calculate Symmetries of Petri nets. *Actae Informaticae*, 36(7):545–590, 2000.

[81] K. Schmidt. Integrating Low Level Symmetries into Reachability Analysis. In *Proceesings of TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*, pages 315–330. Springer-Verlag, 2000.

[82] The SMV System. `http://www.cs.cmu.edu/~modelcheck/smv.html`.

[83] The Cadence SMV Model Checker. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv`.

[84] The SPIN Tool. `http://netlib.bell-labs.com/netlib/spin/whatisspin.html`.

[85] P. Starke. Reachability analysis of petri nets using symmetries. *Journal on Syst., Anal., Model., Simul.*, 8:294–303, 1991.

[86] U. Stern and D. Dill. Parallelizing the Murϕ Verifier. In *Prooceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 256–278. Springer-Verlag, 1997.

[87] A. Tokmakoff and J. Billington. An Approach to the Analysis of Interworking Traders. In *Proceedings of ICATPN'99*, volume 1639 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, 1999.

[88] A. Valmari. Error Detection by Reduced Reachability Graph Generation. In *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets*, pages 95–112, 1988.

[89] A. Valmari. Stubborn Sets of Coloured Petri Nets. In G. Rozenberg, editor, *Proceedings of ICATPN'91*, pages 102–121, 1991.

[90] A. Valmari. The State Explosion Problem. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.

[91] A. Valmari and I. Kokkarinen. Unbounded verification results by finite-state compositional techniques: 10^{any} states and beyond. In *Proceedings of the 1998 Conference on Application of Concurrency to System Design*. IEEE Computer Society, 1998.

[92] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 322–331, 1986.

[93] W. Visser. Memory Efficient State Storage in SPIN. In *Proceedings of SPIN'96 Workshop*, pages 21–36, 1996.

[94] J. Xu and J. Kuusela. Analyzing the Execution Architecture of Mobile Phone Software with Colored Petri Nets. *Software Tools for Technology Transfer*, 2(2):133–143, December 1998.