CHALMERS

Specification of Resource Allocation Systems
a STEP towards a unified framework

PETTER FALKMAN

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2005

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Specification of Resource Allocation Systems

a STEP towards a unified framework

PETTER FALKMAN

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2005

Specification of Resource Allocation Systems
a STEP towards a unified framework

PETTER FALKMAN

ISBN 91-7291-694-X

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 2376
ISSN 0346-718X

Department of Signals and Systems
Automation Research Group
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Telephone +46 (0) 31 772 10 00

Printed by Chalmers Reproservice
Göteborg, Sweden 2005

Typeset by the author with the LaTeX Documentation System

To Kerstin, William and Elsa with love

abstract

Specification of Resource Allocation Systems
a STEP towards a unified framework
Petter Falkman

Department of Signals and Systems
Chalmers University of Technology

In recent years growing demands on flexibility and ability to decrease time to market has made it increasingly important for engineering companies to find ways of making information exchange between product design and manufacturing systems design more efficient. A much shortened iteration cycle could be obtained if information about product design solutions could be made instantly available for engineers involved in manufacturing systems design. Due to the high costs associated with modifying and changing system implementations, the ability to model and simulate systems before they are implemented is becoming more and more essential. Consequently it is vital that the system specifications used are as clear and concise as possible. The present thesis deals with the specification of discrete event systems, especially resource allocation systems. A combination of process algebra and Petri nets is presented. This combination results in a powerful language, called process algebra Petri nets (PPN), for specifying resource allocation systems, delivering both concise and easy-to-read specifications of large complex systems. The fact that both Petri net constructs and algebra expressions can be used in order to decrease specification complexity also makes it a flexible language. A method is also presented that formally converts the PPN models into finite state automata, which means that existing formal evaluation techniques for simulation, verification, and controller synthesis can be easily applied.

The presented language defines an algebra where the process operators express the same process relations that are possible in the international standard STEP-AP214. To the best of our knowledge, the PPN language constitutes a first attempt at using a formal language in order to create a tool that can automatically generate specifications according to the STEP standard.

So far little has been investigated concerning the connection between information modelling and discrete event systems. The present work, however, researches this connection. The presented mapping defines the relationship between the information and the DES specification.

Finally, it can be said that the introduced method guarantees that the expected information is delivered fast and without the errors potentially induced by manual handling, something which is crucial when short lead times are required. Due to the fast information exchange it also enables simulation, automatic controller synthesis and verification, to be conducted early in the development chain.

KEYWORDS: Flexible manufacturing systems, Resource allocation systems, supervisory control theory, supervis synthesis, specification, Petri nets, Process algebra, finite state automata, STEP AP214.

acknowledgements

I would like to thank everyone that in some way or another helped making this work come together. First, my sincerest gratitude of course goes to my supervisor Professor Bengt Lennartson, for introducing me into the world of science so generously sharing his time and knowledge with me, and always believing in me. Second, I wish to thank my co-supervisor Associate Professor Martin Fabian for providing invaluable support throughout the process of writing this thesis. Thank you also to Knut Åkesson for providing valuable comments on earlier drafts of this work.

Special thanks to my collaborators at KTH, especially Johan Nielsen and Astrid von Euler-Chelpin.

I would also like to thank all my friends and colleagues within the department of Control and Automation Laboratory. A special thanks goes to the members of the automation group, for providing moral support and bandying ideas with me whenever I needed it, in particular Kristin Andersson, and Johan Richardsson.

Last, but certainly not least, I am forever indebted to my loving and supporting family for always believing in me: my wife Kerstin and my children William and Elsa, my brothers Pål and Pär, my parents Jan and Marian, and Gavin and Katarina Watson.

Göteborg, November, 2005
Petter Falkman

publications

The thesis is based on the following appended papers:

Falkman, P. and Lennartson, B. and Åkesson, K. and Fabian, M. (2005), A High Level Specification Language based on Process Algebra and Petri Nets, Submitted to: *IEEE Transactions on Automation Science and Engineering*.

Falkman, P. and Lennartson, B. and Åkesson, K. (2005), Formal Specification of Flexible Robot Cell using Process Algebra Petri Nets, Submitted to: *IEEE Transaction on Control System Technology.*

Falkman, P. and Lennartson, B. and Tittus, M. (2005), Specification of a Batch Plant using Process Algebra and Petri Nets, Submitted to: *Control Engineering Practice.*

Falkman, P. and Nielsen, J. and Lennartson, B. (2003), Automatic Generation of Object Models for Process Planning and Control Purposes using an International standard for Information Exchange, In: *Journal of Systemics, Cybernetics and Informatics,* Vol 1, Number 5.

Note: The paper has been reformatted for uniformity, but is otherwise unchanged.

Falkman, P. and Nielsen, J. and Lennartson, B. and Euler-Chelpin, A. (2005), Automated Generation of STEP AP214 models from Discrete Event Systems for Process Planning and Control, Submitted to: *IEEE Transactions on Automation Science and Engineering.*

Other publications

Falkman, P. and Lennartson, B. (2001), Combined Process Algebra and Petri Nets for Specification of Resource Booking Problems, In: *Proc. of the IEEE American Control Conference, Arlington, USA.*

Falkman, P. and Lennartson, B. and Tittus, M. (2001), Modeling and Specification of Discrete Event Systems using Combined Process Algebra, In: *Proc. of the IEEE/ASME Advanced Intelligent Mecatronics, COMO, Italy*.

Falkman, P. and Vahidi, A. and Lennartson, B. (2002), Modelling and controller Synthesis for Resource Booking Problems using BDDs, In: *Proc. of the IFAC 15th World Congress, Barcelona, Spain.*

Falkman, P. and Nielsen, J. and Lennartson, B. (2002), A Formal Mapping of Static Information Models into Dynamic Models for Process Planning and Control Purposes, In: *Proc. of the Workshop on Discrete Event Systems.*

Falkman, P. and Nielsen, J. and Lennartson, B. (2003), Automatic Generation of Object Models for Process Planning and Control Purposes using an International standard for Information Exchange, In: *Proc. of the Systemics, Cybernetics and Informatics, Orlando, Florida, USA.*

Falkman, P. and Lennartson, B. and Tittus, M. (2005), Specification of a Batch Plant using Process Algebra and Petri Nets, In: *Proc. of the IEEE Conference on Automation Science and Engineering, Edmonton, Canada*.

Lennartson, P. and Fabian, M. and Falkman, P. (2005), Control architecture for flexible production systems, In: *Proc. of the IEEE Conference on Automation Science and Engineering, Edmonton, Canada*.

contents

introductory chapters

chapter i

introduction

A discrete event system (DES) is a system that, at any time, occupies one out of a finite set of states, and changes state at the occurrence of an event. Specifications of such systems are often expressed in text-based natural-language documents. A major drawback with this is ambiguity, it is possible to interpret the natural-language specification in many ways. This may lead to misunderstandings that are costly and/or time-consuming if discovered at a late stage of a project. Consequently, it is very important for a costumer and a supplier to agree on a specification written in such a way that misunderstandings are avoided. A way of trying to avoid this kind of problem is to use a formal specification language.

The reasons for using formal methods can be divided in two main areas (Crow, Vito, Lutz, Roberts, Feather and Kelly 2005). One is the use of formal methods for analytical purposes, e.g. controlling if certain properties are fulfilled. The other is to use these methods for descriptive purposes, such as the clarification of document requirements, high level design, or facilitating communication of requirements.

Using a formal specification language comes with a diversity of advantages. One of the most important is that a formal language will provide everyone involved in a project with the the same view of, for example, what a product will look like and how it is going to be manufactured. In other words it is unambiguous and does not leave room for different interpretations in the way a natural-language document would. Another important advantage is that after everyone has agreed on a specification it may be automatically converted to a format that can be used for verification and synthesis. Especially important is of course that the conversion to a more detailed description can be completely automatic, so that changes or errors will not be introduced by mistake due to the human factor.

In order to really make the best use of a formal specification language and to motivate engineering companies to use formal methods for specification, it is very important that it has the capacity to produce specifications that are concise and easy-to-read.

When it comes to discrete event systems, a number of different formal specification languages have been presented. Some of them are algebraically based, some are based on graphical representations, and still others are expressed by temporal logic.

The term ”process algebra” was coined in 1982 by Bergstra & Klop (Bergstra and Klop 1982). A general process algebra includes processes, events, and operators, which are used to build algebraic expressions describing the discrete behavior of a system, cf (David and Alla 1992, Murata 1989). The communicating sequential processes (CSP) language by Hoare (Hoare 1985) and the calculus of communicating systems (CCS) language by Milner (Milner 1980) are the major languages based on process algebra. These languages have been further developed in e.g. (Degano, DeNicola and Montanari 1987, Olderog 1991, Best, Devillers and Koutny 1998, Brinksma 1995). An example of a language based on temporal logic is given in (Rescher and Urquhart 1971).

The Petri net (Peterson 1981) introduced by Carl Adam Petri in the sixties, finite state automata (Hopcroft and Ullman 1979), and StateChart introduced by Harel (Harel, Pnueli, Schmidt and Sherman 1987) (an extension of finite state automata), have intuitive and precise graphical representations. The main reason for this is the finiteness of the structures; Petri nets can with a finite number of elements even represent an infinite state-space. This is typically not the case with process algebra and temporal logic; only in very special cases are the graphical representations finite, and thus usefully represented graphically. Combinations of the mentioned languages have also been considered. These include for example process algebra and Petri Nets given in e.g. (Best, Devillers and Koutny 2001, Mayr 1997, Best, Devillers and Koutny 2002, Pena and Cortadella 1996, Bloom, Cheng and Dsouza 1997, Jmaiel 2000). A common limitation of the mentioned languages and combinations of them is that their strength lies in formal evaluation techniques rather than in visual strength. As an engineering tool it is however important that a formal specification language also results in a clear and easy-to-read description of what is specified. This will increase the understanding of created specifications and thereby also decrease the risk of making mistakes due to misinterpretations.

The present thesis is focussed on specification of resource allocation systems, such as certain classes of flexible production systems. To the best of our knowledge, despite the many formal specification languages already presented in previous research, there still is a lack of efficient tools producing the kind of specifications we desire for this kind of systems, i.e. specifications which are concise as well as easy-to-read and interpret. The aim of this thesis is therefore to present such a specification language. This language combines the graphical features of Petri nets with the compact expressions of process algebra. Compared to earlier work the focus of this language, called process algebra Petri net (PPN), is to a higher extent on high level constructs, readability, and visual strength. A further aim is to use the created specifications for supervisor synthesis applying supervisory control theory (SCT) (Ramadge and Wonham 1987).

More specifically, a process algebra is suggested where the process operators express the same kind of process relations as in the international standard ISO-10303-214 or STEP-AP214 (*ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles* 1994). STEP-AP214 standardizes the exchange of product, process and resource information. It enables an efficient information exchange between product and manufacturing system design, which is crucial in order to decrease lead times. For this reason it is also beneficial to be able to use the presented tool for creating specifications according to this standard. The operators to be defined in order to automatically generate high level specifications according to the STEP-214 standard (Falkman, Nielsen and Lennartson 2004) are sequence, alternative, arbitrary order (in STEP exclusiveness), synchronization and parallel execution.

The suggested PPN language is a powerful formalism for specifying discrete event systems. It is an extension, generalization and formalization of results presented in (Falkman, Lennartson and Tittus 2001, Falkman and Lennartson 2001). Use of PPN delivers compact and concise models of large complex systems. It is flexible in the sense that both Petri net constructs as well as algebra expressions can be used in order to generate easy-to-read specifications and in that way reduce the complexity for the user. In (Falkman and Lennartson 2005b) a method is presented that formally converts the PPN models into finite state automata, which means that existing formal evaluation techniques for simulation, verification, and supervisor synthesis are easily applied. PPN introduces new operators, and while these are defined it still uses the earlier mentioned languages (i.e. Petri nets and process algebra) as a foundation.

The objective is thus to maintain the Petri net part as is, while creating a suitable process algebra that can be used to realize a compact, yet graphical, specification formalism. The process algebra suggested in this thesis is adapted to the international standard for exchange of product model data STEP. This standard defines a number of operators and the suggested algebra defines the same operators in a way that the specifications match each other. Those who use the STEP standard are thus able to recognize the specifications in the Petri net and process algebra models. The main purposes of defining a new specification formalism are thus the following; first it coincides with the specifications generated by STEP (in other words the same operators are in place in both representations), second, a compact yet graphical representation is wanted in order to achieve clear specifications. The reason for choosing Petri nets and process algebra is that these formal language families are both well known and well accepted. The Petri nets are also well known in manufacturing programming communities, since Sequential Function Charts (SFCs) resemble Petri nets to a great extent. The suggested process algebra is new and therefore an aim in this thesis has also been to keep the algebra part as simple as possible. A further aim is also to use the resulting specification as a base for supervisor synthesis. The synthesis is performed according to the supervisory control theory (SCT) (Ramadge and Wonham 1987), which deals with the interaction between the (controlled) plant and the (computer) supervisor. For this supervisor synthesis we use a program called *Supremica*, where finite state automata (FSA) is the modelling formalism. We therefore show how PPN models are formally converted into finite state automata.

1.1 Main contributions

The main contributions of this thesis are the following:

- A high level specification language that combines the compactness of process algebra with the graphical features of Petri net constructs, resulting in a language which can deliver concise and easy-to-read specifications of complex systems. The suggested specification language has, in the present work, been used for resource allocation systems but may be used for other applications as well.

- A formal method for converting the high level specifications into finite state automata. This is significant firstly because using a formal method for conversion limits the risk of introducing manual errors, and secondly because it means that existing formal evaluation techniques for simulation, verification, and supervisor synthesis are easily applied.

- A method for enabling mapping between the information models according to the well accepted international standard ISO10303-214 and discrete event system specifications. The presented mapping has also defined the relationship between the information and the DES specification. This method implies a reliable framework for the exchange of control-related information involving resource, product and process information.

- The high level specification language is evaluated for specification of nontrivial industrial plants, including both manufacturing systems and chemical batch plants.

chapter ii

discrete event systems

A discrete event system (DES) can be defined as a system that, at any time occupies one out of a finite set of states, and that changes state at the occurrence of an event. In a research context it is assumed that events are instantaneous.

Discrete event systems are of great importance both from a practical and a theoretical point of view, and pose a wide range of new intellectual challenges (Heymann and Meyer 1991). There are several existing formal languages that can be used to model discrete event systems. The most common are Petri nets (Reisig 1985, Murata 1989), finite state automata (Hopcroft and Ullman 1979), and process algebra (Hoare 1985, Milner 1989). Petri nets and finite state automata have a graphical representation that is easy to understand, while process algebra is a purely symbolic formal language.

A graphically represented specification is often helpful when trying to create something that can be easily interpreted by many different interested parties, e.g. different departments within a company, or a supplier and a costumer, as they tend to give an intuitive representation of a discrete event system. When specifying larger systems, however, the specifications tend to become unwieldy and the advantage of a purely graphical language decreases. The fact that these languages, both graphical and algebraic, are formal makes it possible to perform analysis of systems. These languages have been around for a long time, mainly within computer science and are foremost used for verification of computer systems. The aim of this chapter is to provide a brief introduction to the four most common formal modelling languages mentioned above.

2.1 Finite state automata

Finite state automata (FSA) (Minsky 1989, Hopcroft and Ullman 1979, Hopcroft, Motwani and Ullman 2001) is one of the most popular languages for describing the behavior of discrete event systems. FSA is a mathematically well defined formalism for modelling discrete event systems. They provide a language for both describing and manipulating these systems when synchronous descriptions are considered. The finite state automata language is built up of states and transitions. The transitions connect the states, and an event is associated with each transition. The occurrence of an event is thus associated with a state change. A finite state automaton models systems with a finite number of states and events, and provides a lucid representation of discrete event systems.

Formally, a finite state automaton (FSA) has the following structure

$$A = (Q, \Sigma, \delta, s, F, M) \tag{2.1}$$

where

- Q is a finite set of states;
- Σ is the alphabet, a finite set of event labels;
- $\delta : Q \times \Sigma \rightarrow Q$ is the partial transition function;
- $s \in Q$ is the initial state;
- $M \subseteq Q$; elements of M are called *accept* or *marked states*;

The set Q is the complete set of states that an automaton can visit. The alphabet Σ, on the other hand, declares all events that an automaton can in any way participate in. Some of the events in the alphabet may never be executed, something that is important when performing supervisor synthesis, described in Chapter 3. The transition function δ declares all transitions from a state via an event to another state. The initial state is given by s. The marked states of an automaton specify that some significant sub-tasks of an automaton have been fulfilled.

2.1.1 Composition

The ability to reason about systems where a number of finite state automata interact with each other is important when working with discrete event systems in general, and with supervisor synthesis in particular, see Chapter 3, especially since finite state automata do not allow explicitly modelling parallel behavior. To model the interaction between automata, the full synchronous composition (FSC) concept from Hoare (1985) is used, see section 2.3.1. The synchronous composition of two finite state automata P and R is a new automaton in which a common event is defined from a certain state <p,r> if, and only if, both automata define the event from their respective states $p \in Q_Q$ and $q \in Q_R$.

Example 1 – Finite state automata The graphical parts of two automata are shown in Fig. 2.1. States are often modelled with circles, and arrows are used to connect the different states. Each arrow is labelled with the name of the event that connects the specific states. A marked state has an extra circle. The initial state has a short arrow pointing at it.

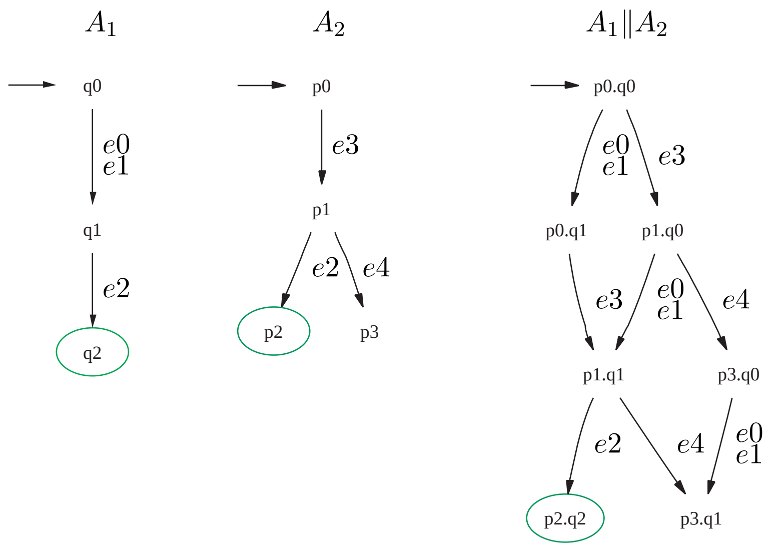


Figure 2.1: Two finite state automata with three and four states respectively and with one common event $e2$.

The two automata, A_1 and A_2, in Fig. 2.1 have three and four states respectively. Automaton A_1 has $Q_1 = \{q0, q1, q2\}$ and automaton A_2 has $Q_2 = \{p0, p1, p2, p3\}$. The alphabet for A_1 is $\Sigma = \{e_0, e_1, e_2\}$ and for A_2 it is $\Sigma = \{e_2, e_3, e_4, e_5\}$. Automaton A_1 has an initial state $s_1 = q0$ and A_2 an initial state $s_2 = p0$. A transition between for instance q_0 and q_1 is described as $<< q_0, e_0 >, q_1 >$. Both automata have one state specified as marked, ie. $M_1 = \{q2\}$ and $M_2 = \{p2\}$, specifying successful completion. The third automaton $A_1 \| A_2$ describes the synchronous composition of the two automata A_1 and A_2. This synchronous composition specifies the parallel behavior of the two automata. They are here synchronized with respect to common events using Hoare's full synchronous composition (Hoare 1985), described in more detail in Section 2.3.1.

□

2.2 Petri nets

Petri nets (Peterson 1981), a formal language used for the modelling of many different kinds of systems, was introduced already in 1962 by Carl Adam Petri. Since it was first introduced, Petri net theory has been extended in a number of different ways and applied to a wide variety of problems. The popularity of Petri net theory is largely based on two factors. First, its easy to understand graphical representation of nets, and second, its potential as a technique for formally analyzing concurrent systems.

One of the aims of Petri net theory is to model concurrent systems and thereby allowing us to reason about them formally. Petri nets can be used as a graphical as well as a mathematical tool. As a graphical tool, Petri nets may be used as a visual communication aid similar to flow charts, block diagrams and graph charts. As a mathematical tool, it enables the setup of state equations, algebraic equations and other mathematical models governing the behavior of different systems.

A Petri net is a bipartite graph consisting of places and transitions. Arcs are used to connect the places to the transitions and vise versa. Each place can be connected to more than one transition and a single transition can also connect to more than one place. Tokens are used in Petri nets in order to simulate the behavior of dynamic and concurrent systems. A place may contain tokens that move from one place to another according to the firing of the transition. Arcs connecting a place with a transition can also have weights on them, which define how many tokens or markings that are required in order for the transition to fire. Arcs connecting a transition with a place on the other hand have weights to define how many markings that are to be positioned in the place when the transition has been fired. Just as in finite state automata it is possible to specify marked states in a Petri net (Cassandras and Lafortune 1999). This is achieved by the use of a marked state vector.

Different kinds of Petri nets exist, e.g. labelled, safe, colored, timed, interpreted, stochastic, continuous. The class of Petri nets used in this thesis are simply labelled Petri nets. Such a broad term can have several meanings, and must therefore be defined more precisely. In the present work, a labelled net will be a tuple

$$PN = (P, T, A, \Sigma, m_0, M) \tag{2.2}$$

where

- P is a set of places;

- T is the set of transitions;
- $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs from places to transitions and from transitions to places in the graph;
- m_0 is a vector of initial markings, specifying the initial number of tokens in each place;
- Σ is a finite set of transition labels that correspond to the events;
- $M \subseteq \mathbb{N}^n$ is the set of marked state vectors;

The marked states, in this definition given by the set M is completely analogous with the notion of marked states in the definition of finite state automata in Section 2.1.

Example 2 – Petri net An example of a labelled Petri net is given in Fig. 2.2. This example models the same system as in Example 1 and involves six Petri net places and five transitions with labels. The initial state is given by the initial marking $m_0 =< 1, 0, 0, 0, 0, 0 >$. The Petri net is safe, i.e. no place can, at any moment, have more than one token. Labels corresponding to the event names are associated with each transition. The Petri net specifies an alternative between event e_0 at transition t_2 and event e_1 at transition t_3, that can occur in parallel with e_3 at t_4. A token in both place $p3$ and $p4$ enables transition t_5 and event e_2. A token in $p4$ enables transition t_6 and event e_4. The marked state in the example symbolizes a successful completion and is defined by a marked state vector $M =< 1, 1, 0, 0, 0, 0 >$. .

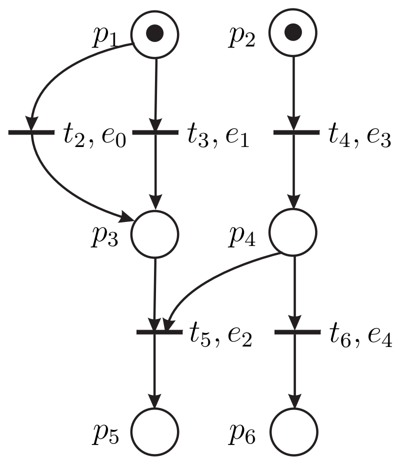


Figure 2.2: A Petri net with five transitions and six places.

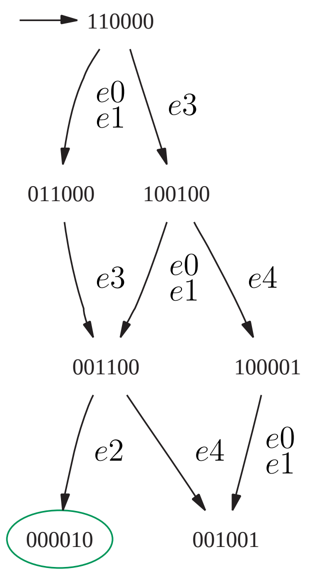


Figure 2.3: The reachability graph for the Petri net in Fig. 2.2.

□

2.2.1 Reachability graph

A Petri net place p_i is said to be *k-bounded* for an initial marking m_0 if there is an integer k so that, for any marking reachable from m_0, the number of tokens in p_i is no greater than k. A Petri net is *bounded* for an initial marking m_0 if all the places are bounded for m_0. This thesis uses so-called safe Petri nets, which are bounded Petri nets where k is equal to one.

When a Petri net is bounded it can be converted into a finite state automaton by the creation of the so-called *reachability graph*. More specifically, the reachability graph is accomplished by writing down the markings resulting from the firing of the transitions starting from the initial marking. The Petri net presented in Example 2.2 has the reachability graph presented in Fig. 2.3. The initial marking in the Petri net in Fig. 2.2 corresponds to the initial state of the reachability graph in Fig. 2.3. The created reachability graph is the same as the finite state automaton describing the synchronous behavior between A_1 and A_2 given in Fig. 2.1.

2.3 Process algebra

When Bergstra & Klop first coined the term *process algebra* in 1982 (Bergstra and Klop 1982) it referred to a structure in the sense of universal algebra that satisfied a particular set of axioms. Since 1984, however, they also used the term to denote an area of science (in this case the term was used as a noun without particle). This means that they sometimes used the phrase to refer to their own algebraic approach to the study of concurrent processes, the so-called algebra of communicating processes (ACP) (Bergstra and Klop 1984, Bergstra and Klop 1985), and sometimes as a referral to such algebraic approaches in general. From this, process algebra has now come to mean mathematical theories that model concurrent systems by their algebra and provide a platform for reasoning about the structure and behavior of a model.

Today, process algebra is the term commonly used to denote an algebra with sequential and parallel operators plus recursion. The word ”algebraic” implies the ability to build a new process description from already existing ones. This means that based on two processes P and Q it is possible to build a third, R, by combining them (building a so-called algebraic structure). All specification languages referred to as process algebra have in common that they define and use operators to describe relations between events and processes, and vice versa. They are often used for the analysis of concurrent system behavior, and particularly for the comparison of different behaviors. The two most well known process algebras are Hoares communicating sequential processes (CSP) (Hoare 1985) and Millners calculus of communicating systems (CCS) (Milner 1989). In this section a short description of both these process algebras is given.

2.3.1 Communicating sequential processes, CSP

A brief description of the communicating sequential processes language (Hoare 1985) is provided here. The aim is to show the most common operators that are used to define the basic CSP algebra and also to relate to finite state automata and Petri nets.

Every process in CSP has an alphabet Σ. The alphabet is a set of all events that the process (and any other related process) might use.

Sequence: The sequence operator can be divided into two different types, the first is sequence between an event on the left hand side of the operator and a process on the right hand side. The process $a{\rightarrow}P$ first performs an event a and subsequently behaves like process P. The second type of sequence operator is the one between processes denoted $P_1; P_2$, which specifies that after successful termination of process P_1 it behaves as process P_2.

Alternative: There are three different operators for specifying an alternative in the CSP language. The first is the *deterministic choice symbol* written as $(a.P|b.Q)$. This choice operator is not fundamental, and non-essential operators like this are used in CSP to add clarity. This operator allows the environment to choose between distinct events. The two events a and b must be uniquely identifiable by the environment. There is no casuality, i.e. it is not possible to say whether the process or its environment "caused" a certain event.

The CSP formalism include a special operator for non-deterministic choice. The *non-deterministic choice operator* $P \sqcap Q$ or $a.P \sqcap b.Q$ defines that it is up to the system to choose which process to execute.

The third is a *general choice operator* $P\square Q$. The environment can control which of P and Q that will be selected, provided that this control is exercised on the very first action. If P can engage in a specific event then it will, irrespective of Q and vice versa. If both P and Q are able to engage in a specific event, then were are faced with a non-deterministic choice.

Full synchronous composition: The second is the fully dependant composition called full synchronous composition (FSC) denoted $P_1 || P_2$. This operator specifies that events common to both processes have to occur simultaneously. This means that which processes that may interact are implicit in the process alphabets. An interaction is visible to the environment and this is necessary in order to support multidirectional interaction. A multi-party synchronization allows a number of processes to interact with a single, shared event.

Concealment: The concealment operator $P\backslash\{e_1, \ldots, e_n\}$ allows the user to explicitly list those events that may occur invisibly, without the participation of the environment. If P is ready to perform some event e_i in the list of concealed events then it may do so immediately, without the need to interact with any other process.

Renaming: A renaming function f is introduced in order to be able to reuse a specification by only replacing one or more of the actions names. A process $f(P)$ can replace one or more events with new event names.

Definitions and recursion: The CSP language defines a way of associating a name with a specified behavior $N = P$. These definitions may be recursive.

STOP process: The stop process $STOP_A$ has an alphabet A and can therefore block other processes when composing the $STOP_A$ with other processes with the same events as in A.

Successful termination: A successful termination operator $SKIP$ terminates immediately and denotes successful completion of some task.

Example 3 – CSP The same example already presented as finite state automata in Example 1 and as a Petri net in Example 2 is here given in the CSP formalism. Note that it is not possible to specify the marked states in CSP. The specification is divided in to three sub-processes, P_1, P_2, and P_3, together with a main process P. Process P_1 in (2.3) specifies the alternative of performing either event e_0 or e_1. Either of these two events are followed by process P_2, which performs event e_2 and then terminates. Process P_3 first performs event e_3 and then either event e_2 followed by $STOP_A$ or event e_4 followed by $STOP_A$ where $A = \{e_0, e_1, e_2, e_3, e_4\}$. The main process P specifies that the processes P_1 and P_2 are to be executed in parallel using the full synchronous composition operator $\|$, which specifies that common events are to be synchronized in the same way as in the Petri net in Fig. 2.2.

$$\begin{aligned}
P_1 &= e_0{\rightarrow}P_2|e_1{\rightarrow}P_2 \\
P_2 &= e_2{\rightarrow}STOP_A \\
P_3 &= e_3{\rightarrow}(e_2{\rightarrow}STOP_A|e_4{\rightarrow}STOP_A) \\
P &= P_1\|P_3
\end{aligned} \tag{2.3}$$

□

2.3.2 Calculus of communicating systems CCS

In this section a short introduction to the calculus of communicating systems (CCS) (Milner 1989) is presented. The aim is to show the most common operators that are used to define the basic CCS algebra and also to relate to finite state automata and Petri net.

Sequence operator: The most basic process construction in CCS is the action prefixing (sequence). This sequence operator is, as in the CSP, between an event and a process as $a.P$. After event a occurs it will behave as an process P.

Choice operator: In order to be able to define processes whose behavior may follow different patterns of interactions with their environment, CCS offers the choice operator, denoted ”$+$”. The formation rule for choice states that if P and Q are processes, then so is $P + Q$. The CCS also provides an indexed choice operator $\Sigma_{i \in I} P_i$.

Composition operator: In order to describe systems consisting of two or more processes running in parallel, and possibly interacting with each other, CCS offers the parallel composition ”$|$”. Given two CCS expressions P and Q, the process $P|Q$ describes a system in which P and Q may proceed independently of each other, and may communicate via complementary parts. In order to specify interleaving using this operator it is required that the two processes have no ”co-names”. An interaction between two processes is immediately hidden from the environment. If two processes interact, this shared event can never be seen by any other processes. CCS processes synchronize only on events with complementary names, e.g. a and $\overline{a}$. It is therefore not possible to express multi-party interaction.

Concealment: The concealment, or restriction, operator $P \backslash \{a_1, \ldots, a_n\}$ in CCS is syntactically identical with the concealment operator in CSP. However, they have very different meanings. The restriction operator in CCS forces a process to interact with its environment. If an event a_i is in the list then it can not visibly occur and only the interaction between two processes can take place, ie. a_i and $\overline{a_i}$.

Internal event: Internal events result from process interaction. A special notation τ is used to specify an internal event, which can be used to specify local behavior.

Renaming: A renaming function f is introduced in order to be able to reuse a specification by only replacing one or more of the event names. Wherever a process is able to perform an event a, process $P[f]$ can perform event $f(a)$.

Definition and recursion: The CCS formalism defines a way of associating a name with a specified behavior $N \overset{def}{=} P$. By introducing names for processes it provides a possibility to define recursive process behavior.

Inaction: The most basic process is the process 0 (nil), which performs no event. This process offers the prototypical example of a deadlock behavior (i.e. one that can not proceed any further in its computation). Milners nil process does not include an alphabet since it is defined as $0 = \Sigma_{i \in \emptyset} P_i$.

Example 4 – CCS The same example already presented as finite state automata in Example 1 and as a Petri net in Example 2 is here specified using the CSP language. Note that it is not possible to specify the marked states in CCS. The example is divided in to three sub-processes, P_1, P_2, and P_3, together with a main process P. Process P_1 in (2.4) specifies the alternative of performing event e_0 or e_2. Either of these two events are followed by Process P_2, which performs event e_2 and then finishes. Process P_3 first performs event e_3 and then either executes event e_4 or executes the complementary action $\overline{e_2}$ before ending. The main process P specifies that the processes P_1 and P_2 are to be executed in parallel using the parallel operator. The restriction expression is used in order to force events e_2 and $\overline{e_2}$ to be synchronized in the same way as in the Petri net in Fig. 2.2.

$$\begin{aligned} P_1 &\overset{def}{=} e_0.P_2 + e_1.P_2 \\ P_2 &\overset{def}{=} \overline{e_2}.0 \\ P_3 &\overset{def}{=} e_3.(e_2.0 + e_4.0) \\ P &\overset{def}{=} P_1 \backslash \{e_2\} \;\mid\; P_3 \backslash \{\overline{e_2}\} \end{aligned} \tag{2.4}$$

□

2.4 Summary

All four formalisms described in this chapter, i.e, finite state automata, Petri nets, as well as the process algebras (CSP and CCS), can be used for the modelling and specification of discrete event systems. It is, however, important to keep in mind that all of them also have limitations that make them less suitable for the creation of specifications for communication purposes when dealing with larger, more complex, systems.

Focus in the present thesis is on supervisor synthesis of resource allocation systems, which will be described in the next chapter. For this type of system, which tends to become large and complex it is very important that specifications are easy to interpret. The present thesis therefore aims at presenting a language that can provide such specifications by combining the compactness of process algebra with the graphical benefits of Petri nets. This language, called PPN, will be presented in more detail in Chapter 4.

chapter iii

supervisory control theory and resource allocation systems

This chapter gives a brief introduction to both the supervisory control theory (SCT) (Ramadge and Wonham 1989) and resource allocation systems. Supervisory control theory is used in order to automatically generate a supervisor for DES, such as resource allocation systems.

3.1 Supervisory Control Theory

The Supervisory Control Theory developed by Ramadge and Wonham (1987) is a general approach for the control of discrete event systems. Using this theory, automatic synthesis of supervisors for discrete event systems can be achieved, given a model of a plant and a specification. The theory divides the events that a plant can execute into two main groups, i.e events that are controllable and those that are uncontrollable. A synthesized supervisor is only able to prevent the plant from executing controllable events. In order to use the supervisory control theory in practice, on larger industry systems, the use of computer tools is required since the number of states in these systems tend to become very large. An example of such a tool is *Supremica* (Åkesson, Fabian, Flordal and Vahidi 2003), which uses a modular approach to supervisory control theory. The use of supervisory control theory also places large demands on the language used to model a plant, as well as on the specifications, i.e they have to be mathematically well founded. The models of both plant and specification will, in the present thesis, be given as finite state automata when applying the supervisory control theory.

3.1.1 Plant

The plant that is to be controlled is modelled as a discrete event system describing all behavior that the system is capable of, the so-called uncontrolled behavior. This model should describe everything that the plant can do. It is, however, important not to make the model more detailed than is necessary. If the model becomes too detailed this will only result in a large system with unnecessary states, making the calculations harder. At the same time it is of course also important to have a detailed enough model of the plant. A plant model should not include anything that can be interpreted as specification.

It is not self-evident how to model a plant, i.e. which level of detail that is required, which languages should be used for the modelling and so on. Finite state automata are typically used, but very little modelling techniques to arrive at a good model exist in the literature. Ongoing research aims at creating such methods and has also presented a language called *sensor activation graphs* (SAG) (Alenljung and Lennartson 2005), which focuses on the modelling of the plant.

In Fig. 3.1 a model of a plant is presented. It consists of two finite state automata A_{p1} and A_{p2}, which describe the uncontrolled behavior of the plant. A complete model of the plant is achieved by composing all automata models in the way described in 2.1, ie. $A_{p1}||A_{p2}$, see Fig. 3.1.

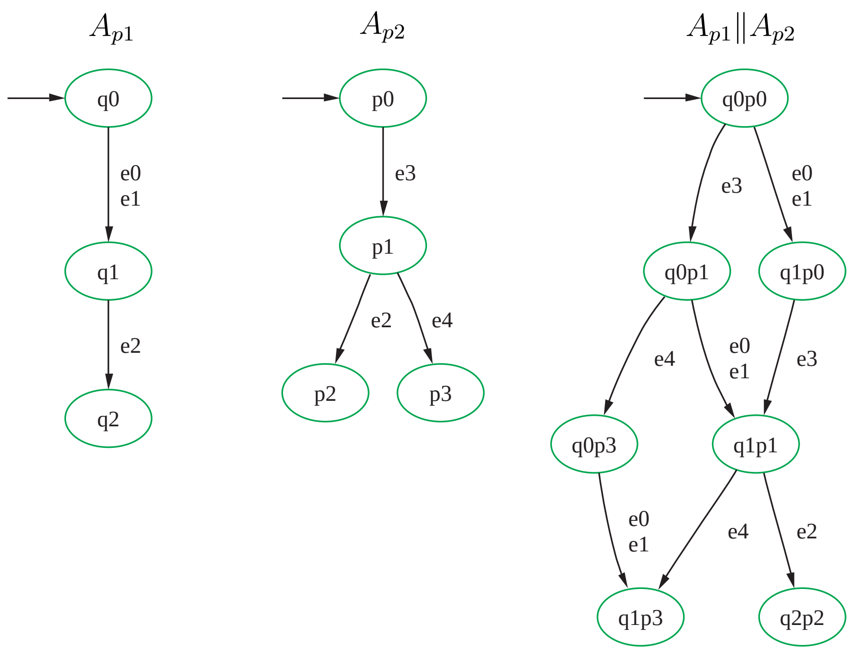


Figure 3.1: Two plant models describing the uncontrolled behavior of the plant.

3.1.2 Specification

A specification describing what the system is intended to do, rather than what it is capable of doing, is also modelled as a finite state automaton. This specification can be partial or total. The alphabet of a partial specification is a subset of the alphabet of the plant. A total specification has the same alphabet as the plant. A partial specification is often used to describe a smaller part of the desired, or undesired, behavior of the plant. A total specification can be achieved by composing the plant model and the specification using the full synchronous composition. In the specification it is possible to specify a state as marked. A marked state specifies that the particular state is significant in some sense, e.g. that some sub-task has been fulfilled. The marked states are used in the supervisory control theory in order to find blocking states, which are states that can never reach a marked state. A partial specification Sp is in Fig. 3.2 given for the plant in Fig. 3.1. This specification specifies that after event $e0$ has occurred event $e2$ is desired, which is specified by $e2$ reaching the marked state $s2$.

3.1.3 Supervisor synthesis

The supervisory control theory describes how a supervisor may be automatically synthesized so that the closed loop system stays within the given specification while at the same time making sure that it does not put too many limitations on the plant.

As a first step towards obtaining a supervisor, the plant models are synchronized with the partial specifications. Assume that m specifications $Sp_1, \ldots, Sp_m$ and n plant models $R_1, \ldots, R_n$ are given. Then the model

$$Sp = Sp_1 || \ldots || Sp_m \tag{3.1}$$

is a *specification* of the desired behavior of the plant

$$P = P_1 || \ldots || P_n \tag{3.2}$$

The two models P and Sp together constitute what is called the *global specification* $S_0 = P||Sp$. This is a first candidate for a possible supervisor S. This model may, however, be blocking, (Ramadge and Wonham 1987). The global specification S_0 therefore has to be manipulated in some way in order to result in an appropriate supervisor. This is formally expressed by the operator $\mathcal{NB}$, which removes blocking states from S_0 to make it trim. The synthesized supervisor is now expressed as

$$S = sup\mathcal{NB}(S_0) = \mathcal{NB}(P_1 || \ldots || P_n || Sp_1 || \ldots || Sp_m) \tag{3.3}$$

The partial specification in Fig. 3.2 is synchronized with the plant model of Fig 3.1 in order to obtain a total specification S_0 in Fig 3.3, which also constitutes a first supervisor candidate. Blocking states are then detected when performing the supervisor synthesis, resulting in the supervisor S in Fig. 3.4, for the plant which guarantees the specification Sp.

$$S = sup\mathcal{NB}(S_0) = sup\mathcal{NB}(A_{P_1} || A_{P_1} || Sp) \tag{3.4}$$

3.2 Resource allocation systems

The present paper is focused on specification of resource allocation systems, such as certain classes of flexible production systems. In this section the different building blocks of this type of resource allocation system are described. These building blocks are made up of a set of resource models, a set of product specifications, and finally a supervisor that synchronizes the individual product's use of shared resources.

Routing and resource allocation systems may be described by a set of shared resources and a set of products. The products use the set of resources in order to be manipulated according to a certain product specification. This specification consists of a set of operations that are to be executed in a certain order by specific resources. The first operation has to be performed by one resource and the second operation by another resource etc. This results in a desired product route through the resource system, and hence the product specification is also called a *routing specification*.

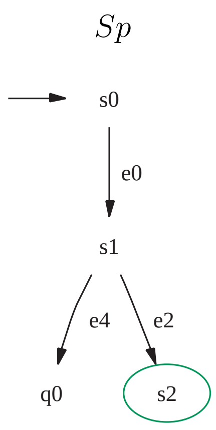


Figure 3.2: A specification describing the desired behavior.

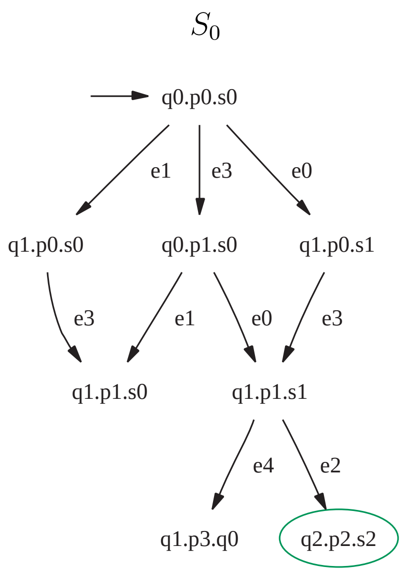


Figure 3.3: The synchronous behavior of the three models, ie. P_1, P_2, and Sp.

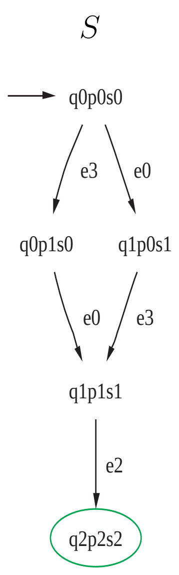


Figure 3.4: The synthesized supervisor S.

3.2.1 Routing specification

Every product to be manipulated has its own route through a system. This route is specified by a routing specification S_i, which may be described on two levels:

- a *high level routing specification*, describes which operations a product are to undergo, in which order these operations are to be executed, and which resource(s) that may be used for each individual operation.

- a *booking and unbooking specification*, describes on a more detailed level how the shared resources are to be booked and unbooked, based on the HRS, in order to obtain the desired route through the resource system.

Information necessary for specifying a high level routing specification for a product can be divided into three different parts (Richardsson 2005). The first part is the so-called *relation of operations* (ROP), which describes on a high level operation sequences, predecessors and required resources for each operation. This information is created when deciding the necessary order of operations to meet the demands of the product design, e.g. a geometrical welding must always precede a weld spot welding operation in order to guarantee a correct design of the finished product. The second part is the execution of operations (EOP), which describes on a detailed level the sequence of events for the specific operation. This is described by specifying the resources that will change state on a certain event. This means that an EOP is specified by a number of resources changing states. The third, and last, part is the interlocking (IL). This describes restrictions on specific events in the EOP. These can be, for example, security restrictions, i.e. that no person is allowed inside a restricted area when a robot is to move, or restrictions specified in order to avoid equipment collisions.

Since both the EOP and the IL are described on a more detailed level than the ROP, a so-called specification synthesis (Andersson, Richarsson, Lennartsson and Fabian 2005) is performed. In this synthesis the EOP and the IL are compared and, the result is an allowed sequence of operations taking into account the possible interlocks. The specification synthesis will result in a safety specification, which together with the ROP represents the complete high level routing specification for a product.

3.2.2 Resource

A model of the manufacturing system, considered as a resource allocation system, is created by synchronizing all of the involved resource models. The different resources may be modelled as two state models with two events, the booking and the unbooking event. In order to achieve a deterministic resource allocation model, however, a specific place for each product is included as in Example 5.

Example 5 – Resource and routing specifications Resources are either booked b_ℓ^i or unbooked u_ℓ^i where the indexes ℓ and i refer to the concerned resource, and the routing specification using it. This can be expressed as a PN as in Fig. 3.5. The use of resource R_ℓ is therefore controlled by the resource model, which models the mutual exclusion between the routing specifications using it.

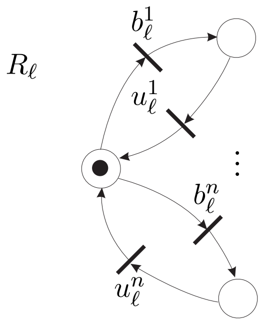


Figure 3.5: A resource model with explicit places for each routing specification.

□

3.2.3 Synthesis

In order to synchronize the different products use of the available shared resources, a *supervisor* is required. This supervisor, which may be thought of as an intelligent booking procedure, is automatically constructed and adapted to the current resource/routing information. From a user point-of-view the basic idea is that the products are to route themselves through the resource system. In this respect the purpose of the supervisor is simply to prevent products from visiting undesired states; states leading to forbidden states such as deadlocks.

Example 6 – Booking and unbooking specification In this example a booking and unbooking specification is described in order to show the main principle of a resource allocation system. In Fig. 3.6 there are two products to be manipulated based on the booking and unbooking specifications, S_1 and S_2. S_1 requires resource R_1 and then R_2, and S_2 requires the same resources but in the opposite order. The resources are modelled as in Fig. 3.5. The second resource in each specification is booked before the first resource is unbooked. A total specification S_0 and a first candidate for a supervisor $S_0 = R_1||R_2||S_1||S_2$ is given as a safe Petri net in Fig. 3.7.

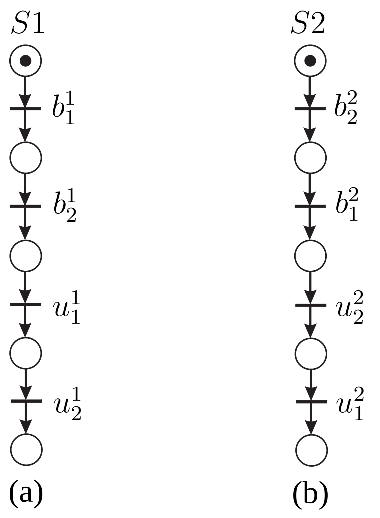


Figure 3.6: Routing specifications S_1 and S_2 given as booking/unbooking models.

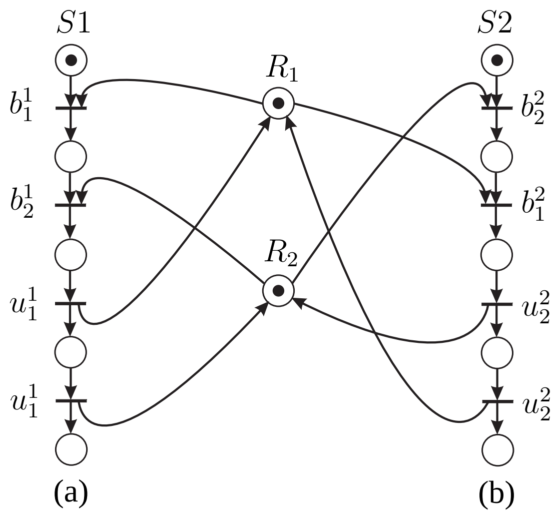


Figure 3.7: The synchronization of the resource models and the routing specifications, i.e. the global specification S_0 as a Petri net.

An automaton (the reachability graph) corresponding to the total specification in Fig. 3.7 is shown in Fig. 3.8, where the blocking state is indicated with a rectangle. This blocking state represents that S_1 has booked resource R_1 and wants to book resource R_2, while S_2 has booked resource R_2 and wants to book resource R_1.

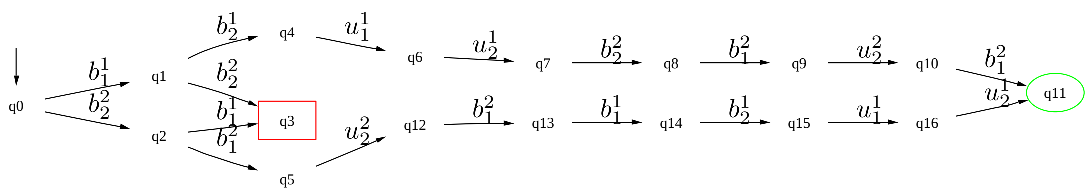


Figure 3.8: The reachability graph of the total specification in Fig. 3.7.

The operator $\mathcal{NB}$ calculates a nonblocking supervisor. A non-blocking supervisor is shown i Fig. 3.9 which guaranties the specifications in Fig. 3.6.

□

$$\begin{tikzcd}
 & \text{q1} \arrow[r, "b_2^1"] & \text{q4} \arrow[r, "u_1^1"] & \text{q6} \arrow[r, "u_2^1"] & \text{q7} \arrow[r, "b_2^2"] & \text{q8} \arrow[r, "b_1^2"] & \text{q9} \arrow[r, "u_2^2"] & \text{q10} \arrow[dr, "b_1^2"] & \\
\arrow[r] & \text{q0} \arrow[ur, "b_1^1"] \arrow[dr, "b_2^2"'] & & & & & & & & \text{q11} \\
 & \text{q2} \arrow[r, "b_1^2"] & \text{q5} \arrow[r, "u_2^2"] & \text{q12} \arrow[r, "b_1^2"] & \text{q13} \arrow[r, "b_1^1"] & \text{q14} \arrow[r, "b_2^1"] & \text{q15} \arrow[r, "u_1^1"] & \text{q16} \arrow[ur, "u_2^1"'] &
\end{tikzcd}$$

Figure 3.9: A non-blocking supervisor, which guaranties the specification in Fig. 3.6.

3.3 Discussion

In the beginning of Section 3.2.1 it was described how a routing specification can be given on two different levels, i.e. as a high level specification and as a booking and unbooking specification. So far, however, only the booking and unbooking specification has been used for the description of the resource allocation system. Ordinary labelled safe Petri nets have been used for the creation of this specification. However, since the high level routing specification involves the ROP, the EOP, and the IL specifications it is necessary that the specification formalism used is able to specify sequences of operations on a high level as well as to represent detailed descriptions of operations and interlock descriptions in a concise and easy-to-read manner.

In the next chapter such a language will be presented. An example of a high level routing specification will also be presented, describing all the necessary specifications, ie. ROP, EOP, and IL, needed to describe a complete routing specification.

An important point in this context is also that a synthesized supervisor is often in the form of a finite state automata and therefore needs to be converted into a format usable by the industrial control system. This is not always straight forward and a lot of research has been carried out within this area. The conversion from finite automata to Sequential Function Charts (SFC), for example, has been treated in detail in (Hellgren 2000).

chapter iv

process algebra petri nets

All formalisms for specification of discrete event systems that have been described so far i.e, finite state automata, Petri nets, and process algebras, may be used in order to specify a resource allocation system. All of them, however, lack specific features that makes them unsuitable for the purpose of creating specifications that fulfill the criteria of being concise and easy to read even when larger systems are specified. Finite state automata, for example, does not explicitly support parallelism and since parallel behavior is very common, this is an essential drawback. Petri nets, on the other hand, explicitly supports parallelism, but when the specification concerns larger systems PNs tend to become large with a lot of interconnections between transitions and places, which makes it difficult to handle and interpret. Process algebra, finally, does support parallelism but, just like Petri net specifications, specifications created using a pure algebraic language become very hard to read if the expressions become too large. One way of avoiding this problem could be to create a lot of smaller processes, but this instead makes it difficult to grasp the complete specification.

As mentioned above, the visual representations typical of Petri nets, although easily interpreted, are not really helpful when dealing with more complex nets. One way of dealing with this problem is to create smaller parts of a specification in a modular fashion and then use existing methods for composing these. This may, however, have a negative effect on readability. In such cases, it could instead be particularly fruitful to study combinations of several methods (Basten 1998), or formalisms with different characteristics that might complement each other. Such studies may also lead to improved and more complete methods and formalisms, as well as a better conceptual understanding, for example, concerning different strengths and weaknesses of the methods and formalisms used.

4.1 Combinations of Petri nets and process algebra

One possible combination is using Petri nets together with process algebra. These two formalisms share important characteristics. First, they both have a precise mathematical definition, and second, they are both designed for reasoning about so-called *concurrent systems*. Many different attempts to join Petri nets and process algebra have been made, for many different reasons. Many of these attempts have concentrated on the translation from one formalism to another in order for both formalisms to be used within the same framework.

The many attempts at combining Petri nets and process algebra include, for example Olderog (1991), who use operational semantics on Petri nets in order to infer process algebra operators. Another example is Rondogiannis and Cheng (1994) who represents process algebra by safe Petri nets and uses Petri net theory to reason about these programs. Mayr (1997) presents a method that extends Petri nets with the possibility to call subroutines, thus making it possible to model recursion and tree-like structures. A methodology for the automatic synthesis of asynchronous circuits from descriptions based on process algebra is presented by Pena and Cortadella (1996). This method combines process algebra and Petri nets in a way that higher tasks are modelled by the use of process algebra, while the Petri nets plays the part of an assembly language, describing the low level operations that are to be executed. Jmaiel (2000) presents a method for specifying a protocol using different formalisms. This protocol offers the possibility of integrating different process descriptions in the same algebraic specifications. Hermanns, Herzog, Mertsiotakis and Rettelbach (1997) has suggested a method that constructs large generalized stochastic Petri nets by a hierarchical composition of smaller components. This is a promising way of coping with the complexity of the design process for models of real hardware and software systems. The composition of nets is inspired by process algebra operators. Basten (1998) combines process algebra and Petri nets for specification and verification by using a method where it is not necessary to use both languages in one and the same model, but rather gives the possibility of using each language separately for different purposes.

The most comprehensive attempt at combining process algebra and Petri nets, however, is perhaps the Petri box calculus (PBC), also referred to as Box algebra. Petri box calculus was developed by Best (Best et al. 2001, Best et al. 2002) in order to show how Petri nets can be manipulated algebraically. This work also shows how Petri net methods can be applied to the verification of concurrent algorithms. Petri box calculus aims at supporting Petri net semantics. It creates an algebra in order to express Petri nets and uses a way of symbolizing Petri net tokens. This can be thought of as an analytical way of combining Petri nets and process algebra. In order to obtain a compositional translation of PBC expressions into Petri nets they define for the latter operators corresponding to those introduced for the process algebra. A uniform translation from algebra expressions to Petri nets is achieved using relabelling, which specifies interface changes applied to nets involved in the transitions.

The present thesis also introduces a combination of Petri nets and process algebra. The single reason for doing so is to create a graphical specification formalism that is able to achieve compact and easy-to-read specifications of complex systems. In other words, it is designed with the visual quality of the resulting specification in mind, rather than the formal evaluation. The specification created is instead formally converted into finite state automata in order to perform simulation, verification, and supervisor synthesis.

As mentioned previously, the focus of this work is on resource allocation systems, and there is today, still no existing tool that can be used to create specifications with the desired qualities for this type of system. For this reason, the present thesis defines a new specification language to be used for such systems. The suggested language is based on process algebra and Petri nets and has been called process algebra petri nets (PPN). The Petri nets are used in order to produce a formal graphical language and process algebra for producing compact descriptions.

As described in Chapter 3, a resource allocation systems involves resource models, routing specifications, and a supervisor. As was described in the previous chapter, a routing specification can be described on two levels, and the focus in this thesis has been to develop a specification language which is well suited for high level routing specifications. The aim is to create high level routing specifications of resource allocations systems, which do not specify explicit booking/unbooking events, cf. Example 6, but instead describe the product route using operation sequences and required resources for each operation. In the following section a description of the PPN language is presented together with an example of a complete high level routing specification with a relation of operation, an execution of operation, and an interlocking specification. For detailed definitions of the suggested process algebra the reader is referred to Paper I.

4.2 PPN language

The PPN language combines Petri nets and process algebra in order to achieve a specification language that delivers concise and easy to read specifications of complex DES, especially resource allocation systems. The algebra expressions are used for compactness and Petri nets for making the PPN language graphical. The suggested language combines Petri nets and process algebra by allowing a process expression at each Petri net transition. In Paper II it is shown how the PPN can be used when specifying a a resource allocation system for a manufacturing system. Paper II also presents a method for converting PPN specifications into a finite state automata representation. It is in Paper III shown how the PPN language can be used for the specification of a smaller batch plant.

4.2.1 Petri nets

In the PPN language ordinary labelled safe Petri nets (PNs) are used. This means that there are no more than one token at each PN place at any time. The PN part of the PPN language may be used to model simple sequences, but can at the same time also model more complex constructions such as parallelism and recursion.

PNs used in the PPN language one *initial* and one *connector* place. This correspond to a PN that start (and end) with a single place marked, i.e. a single place with a token. The initial and the connector places are used when converting the PPN models into PN. This means that the PN models only allow one place with a token in the initial marking vector. This is no restriction since a PN with more than one token in the initial marking vector can always be reconfigured such that only one place has an initial token, see Fig 4.1.

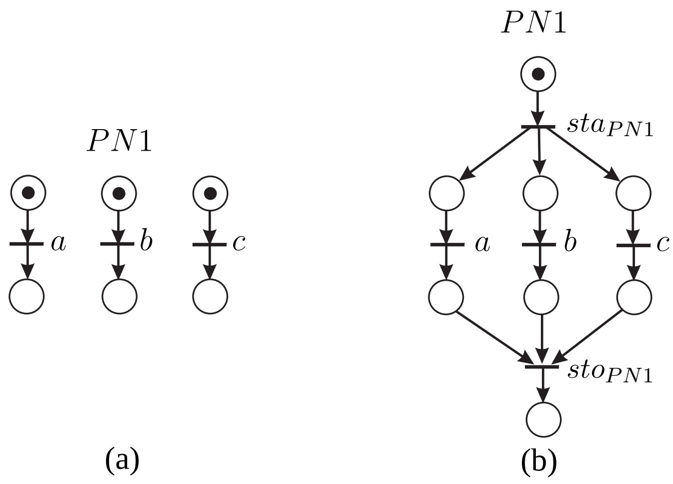


Figure 4.1: Example of how a PN with several initial tokens in several places in (a) is reconfigured such that only one place has an initial token in (b).

The connector place of a PN, in the PPN, describes that a single place then has a token, in much the same manner as for the initial place. The reason for these restrictions is to simplify the translation from PPNs to ordinary labelled safe PNs. The extra transitions that are added in order to realize this are uniquely labelled with a start event sta_i and a stop event sto_i.

4.2.2 Process algebra

A new process algebra is used in the PPN language and the reason for this is, firstly, that there does not exist a process algebra that goes hand in hand with the international standard for information exchange STEP (TC184/SC4 1994), see Chapter 5. This means that operators defined in the STEP standard do not appear in existing process algebras, e.g. the exclusiveness composition. Furthermore, as this thesis aims at creating high level routing specifications of resource allocation systems it is felt that a new synchronization operator is of great importance to keep the specifications clear and concise. Note, however, that the suggested process algebra is a simplified one, using existing process algebra as a foundation, developed to meet our needs in fulfilling the aim of the present work.

The suggested process algebra defines seven operators, $\rightarrow$, $+$, $\oplus$, $\&$, $\Cup$, $P^{\uparrow}$, and $P^{\downarrow}$, which are described in detail in Paper I. The first five operators all involve two processes, while the last two operate on single processes. Below is given a short description of each operator defined for the process algebra part of the PPN language.

Two simple processes $P_1 = a$ and $P_2 = b{\rightarrow}c$ will be used to exemplify the different operators. Corresponding PNs are shown in Fig. 4.2.

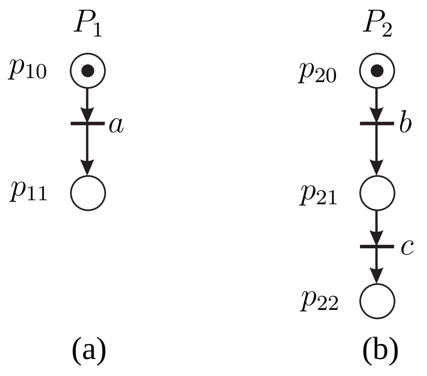


Figure 4.2: Processes $P_1 = a$ and $P_2 = b \rightarrow c$ given as PNs.

Sequence The sequence operator describes that one process has to finish before another process can start to execute. The expression $P_1 \rightarrow P_2$ is a process that begins like P_1 and when P_1 has successfully finished, continues as P_2. This is illustrated in Fig. 4.3 where the two processes from Fig. 4.2 are used to model the sequence $P_1{\rightarrow}P_2$. The transitions in a PPN can involve more than one process as in Fig. 4.3a or only one process at each transition as in Fig. 4.3b. The same sequence is in Fig. 4.3c modelled as a PN with explicit events at each transition.

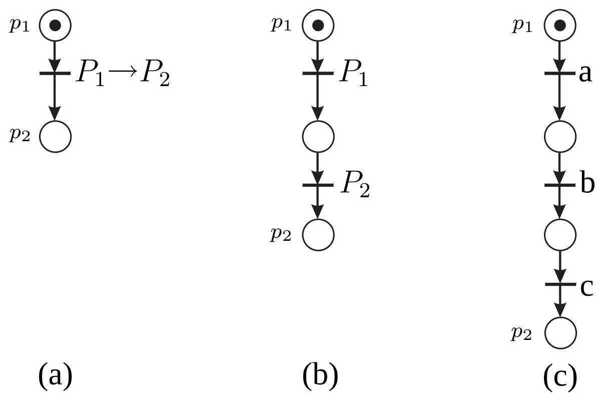


Figure 4.3: The process $P = P_1 \rightarrow P_2$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

Alternative The alternative operator $+$ defines a choice between two processes. The alternative choice between the two processes in Fig. 4.2 is modelled in Fig. 4.4 both as PPNs and PN.

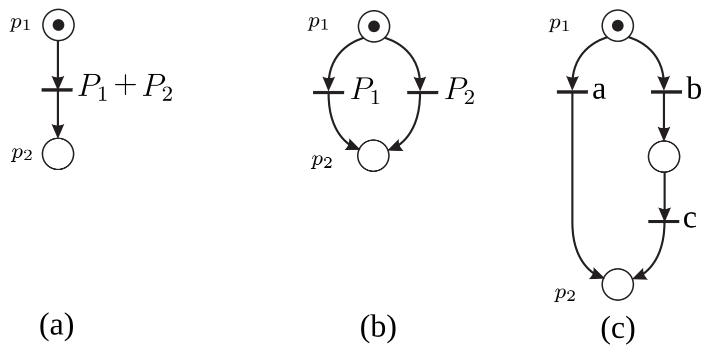


Figure 4.4: The process $P = P_1 + P_2$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

Arbitrary order The arbitrary order operator $\oplus$ defines that a number of processes can not execute at the same time. It does not, however, matter in which order they are executed. The expression $\oplus\{P_1, \ldots, P_n\}$ specifies that n processes are to be executed in any order but never at the same time. This operator is not fundamental, but still necessary in order to create a process algebra that goes hand in hand with the STEP standard described in Paper IV and Paper V. The arbitrary order between the two processes in Fig. 4.2 is demonstrated in Fig. 4.5.

Synchronization A special synchronization operator $\&$ is defined for the PPN language. This operator synchronizes the first and last event of each process. This is very useful when specifying resource allocation systems since it enables the booking and unbooking of a resource to be easily specified. The expression $P_1 \& P_2$ describes that the first event of process

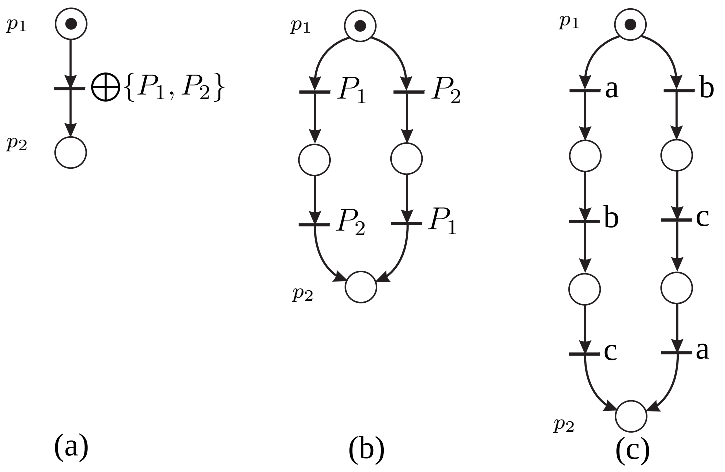


Figure 4.5: The process $P = \bigoplus\{P_1, P_2\}$ is in (a) and (b) given as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \to c$.

P_1 is synchronized with the first event of process P_2, and the same is true for the last events of each process. This means that if P_1 is an operation and P_2 is a resource, then the first event of the operation, e.g. a start event, is synchronized with the booking of the resource and the finishing of the operation is synchronized with the unbooking of the resource.

The explicit synchronization of the two processes described in Fig. 4.2 is demonstrated in Fig. 4.6.

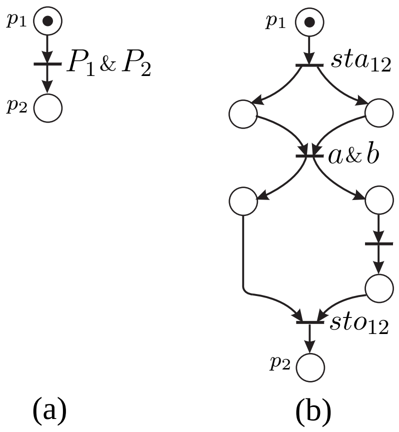


Figure 4.6: The process $P = P_1 \& P_2$ given in (a) and (b) as PPN models using the processes $P_1 = a$ and $P_2 = b \to c$.

Parallel The parallel operator $\Vert\{P_1, \ldots, P_n\}$ specifies that n processes can execute at the same time. It is an extension of Hoare's full synchronous operator in that it can handle synchronized events. Two processes are given as $P_3 = d{\to}e\&f{\to}g$ and $P_4 = h{\to}e{\to}m$ with the common event e. These are to be executed in parallel (as shown in Fig. 4.7).

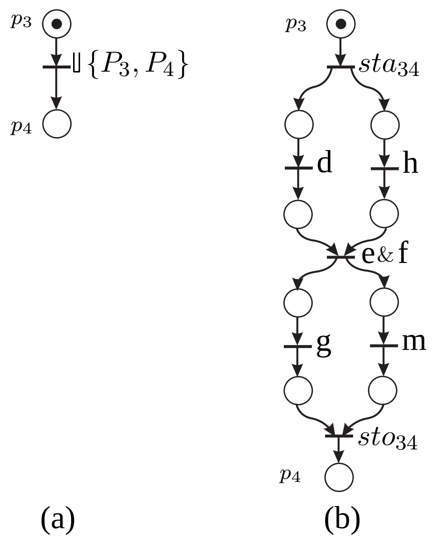


Figure 4.7: The process $P = \Cup\{P_3, P_4\}$, with $B = \{3, 4\}$, given in (a) as a PPN model and in (b) as a PN (including synchronized events).

The synchronized event $e\&f$ in process P_3 is required to occur at the same time as the event e in P_4.

Start and stop operator The start and stop operators, $P^{\uparrow}$ and $P^{\downarrow}$ define that the first and last events respectively are to be executed. In Fig. 4.8 the sequence $P_1{\to}P_2^{\uparrow}$ is illustrated. This expression has the consequence that P_1 executes before P_2 can start. In this case the continuation of process P_2 is not specified, only the fact that it has to wait until P_1 has finished before it is allowed to start.

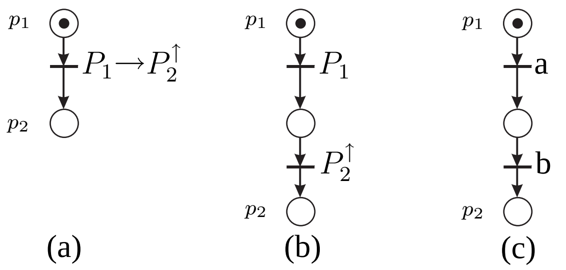


Figure 4.8: The process $P = P_1 \to P_2^{\uparrow}$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \to c$.

Restrictions One of the reasons that it is not very efficient to use a graphical language when specifying larger, and more complex, systems is that the specifications can become hard to interpret due to the many interconnections between different transitions and states. The restriction expression $P[P_1^{\downarrow} \wedge P_2^{\downarrow}]$ specifies that both process P_1 and P_2 have to have finished their respective execution, i.e. be in their respective final states, before process P

is allowed to execute. This means that it is possible to specify that specific states have to be fulfilled, which is what is needed in order to describe both EOP and IL specifications.

4.2.3 Comparison of PPN with CCS, CSP, and StateCharts

In the PPN language the sequence operator, $\rightarrow$, is defined for specifying sequence between processes rather than between an event and a process, or between events. This makes it similar to the operator ; defined in CSP.

The alternative operator $+$ in the PPN language is the same as Milner's choice operator. The synchronization operator $\&$ is different from both the FSC operator $||$ and the $|$ in CCS, since $\&$ neither synchronizes common events (CSP) nor complementary events (CCS).

As mentioned earlier, the parallel operator $\parallel\!\!\mid$ in the PPN language is defined in a way that is very similar to the full synchronous composition operator $||$ used in CSP. A difference between them, however, is that the operator $\parallel\!\!\mid$ is also able to handel synchronous events, e.g. $a\&b$.

StateCharts (Harel et al. 1987) is an extension of finite state automata, which introduces concepts such as event-condition, hiding and hierarchy for finite state automata. The restriction expressions defined for the PPN language are very similar to the condition concept in StateCharts. In Fig.4.9(a) a simple PPN specification is shown.

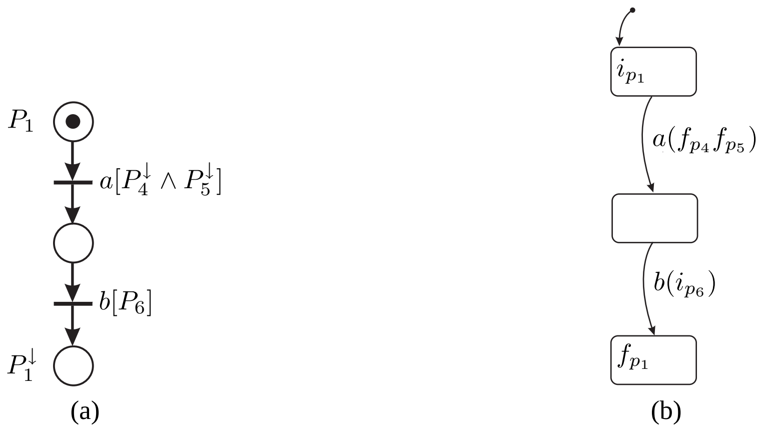


Figure 4.9: A specification with two transitions in (a) given as a PPN model using the restriction constructs and in (b) given as a StateChart using the condition construct.

It involves three places and two transitions. Each transition specifies a restricted event. The first transition specifies that event a cannot occur until processes P_4 and P_5 have finished their respective executions, i.e. are in their respective final states. Event b in the second transition can execute when process P_6 is in its initial state. A corresponding stateChart describing the same specification as the PPN model in Fig. 4.9(a) is given in Fig. 4.9(b). It uses the condition representation in order to specify that event a may not occur until processes P_4 and P_6 are in their respective final states, f_{p_4} and f_{p_5}. A condition also specifies that event b may occur as long as P_6 is in its initial state i_{p_6}.

4.3 Example: Specification of a high level routing specification using PPN

The three parts of a high level routing specification, i.e. the relations of operations, the execution of operations, and the interlocking specifications for a fictive example are specified using the PPN formalism. These three parts are, as described earlier, necessary in order to achieve a complete resource allocation system (Richardsson 2005), and it is in this example shown that the PPN formalism is flexible enough to be able to specify these three models in a way that results in easily interpreted specifications that also play a central role in supervisor synthesis.

4.3.1 Relations of operations specification

This example starts by presenting a relation of operations S_1 in Fig. 4.10 specified using the PPN formalism. The specification constitutes six operations and have a arbitrary order execution that is followed by two parallel paths, which is specified using the parallel PN construct.

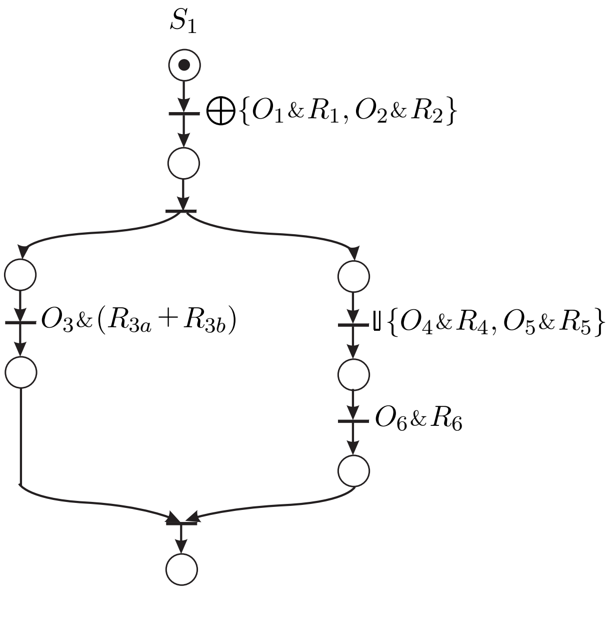

S_1
$\bigoplus\{O_1 \& R_1, O_2 \& R_2\}$
$O_3 \& (R_{3a} + R_{3b})$
$\Vert\{O_4 \& R_4, O_5 \& R_5\}$
$O_6 \& R_6$

Figure 4.10: A ROP given as a PPN.

Operation O_1 requires resource R_1 and operation O_2 requires resource R_2. This is specified using the synchronization operator $\&$, which synchronizes the first and last event of operation O_i with resource R_i. Operations O_1 and O_2 can not execute at the same time, but the specific order in which they execute does not matter. This is specified using the arbitrary order operator $\oplus$. After O_1 and $O2$ have finished their respective execution two parallel paths are created using the PN construct. The first parallel path specifies that operation O_3 can execute. Operation O_3 requires either of resources R_{3a} or R_{3b}, which is specified using the alternative operator $+$. The second parallel path specifies a sequence between the parallel execution of operations O_4 and O_5 and operation O_6. The parallel execution of operations O_4 and O_5 is modelled using the parallel operator $\Vert$.

This simple ROP specification shows how the PPN language can be used in order to specify a high level routing specification in a compact, but yet readable, way. It also shows the flexibility of using either the PN constructs or the process algebra in order to realize the intended specification.

4.3.2 Execution of operations and interlocking specification

In the method presented in (Richardsson 2005) a resource constitutes a number of sub-resources, also called components. These components are used in order to create a detailed description of each operation, or rather the specified states of a component are used to describe an operation. Resource R_6 in this example constitutes two components; R_{6a} and R_{6b}, and both these components are used in order to create the EOP specification. Component R_{6a} can either be in state up or $down$ and component R_{6b} can also be in either of two states on or *off*.

In Fig. 4.11(a) an EOP specification of operation O_6 is presented. This operation describes a sequence of three events $goDown_{R_{6a}}$, $turnOn_{R_{6b}}$, and $finish$. Operation O_6 is described by changes in the components states. The first event $goDown_{R_{6a}}$ is executed when R_{6a} is in state up and R_{6b} is in state *off*. This event is followed by event $turnOn$, which has to wait until R_{6a} has changed state to $down$ before executing. The last event $finish$ is executed when component R_{6b} has changed state to on. An interlock for event $goDown_{R_{6a}}$ is described in Fig. 4.11(b). This interlock specifies that another component R_{2a} has to be in state $home$ before $goDown_{R_{6a}}$ can execute.

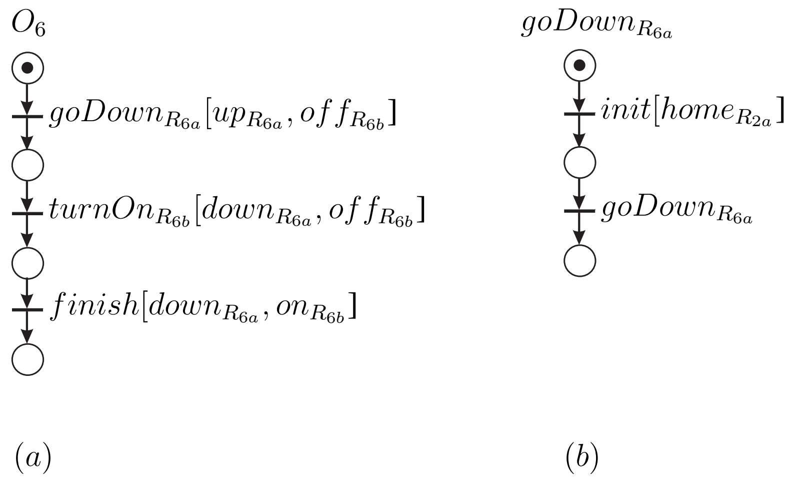

O_6
$goDown_{R_{6a}}[up_{R_{6a}}, off_{R_{6b}}]$
$turnOn_{R_{6b}}[down_{R_{6a}}, off_{R_{6b}}]$
$finish[down_{R_{6a}}, on_{R_{6b}}]$
(a)
$goDown_{R_{6a}}$
$init[home_{R_{2a}}]$
$goDown_{R_{6a}}$
(b)

Figure 4.11: An EOP in (a) describing operation O_1 and in (b) and interlocking for event *turn off*$_{R_{1a}}$.

These more detailed specifications show the generality of the PPN language. It is not only possible to model larger systems on a high level, but it is also possible to create specifications on a lower level using the restriction expressions from the suggested process algebra.

4.3.3 Routing specification

The information given in both the EOP and the IL specifications is then used in a *specification synthesis*. This synthesis creates a so-called *safety specification* that describes, on the same level as in the ROP specifications, i.e. sequence of operations, in which order it is possible to execute the involved operations in order to guarantee the IL specification. This is described in detail in (Andersson et al. 2005). The resulting safety specifications together with the ROP specification specifies the possible order in which the operations for a product can be executed and thus results in a complete high level routing specification.

4.4 Summary

This chapter has shown how the suggested PPN language can be used to create routing specifications on a high, as well as a more detailed, level. A further aim of the present thesis has also been to use the international standard, STEP, in order to be able to communicate the created information between different computer systems and users. In order to make this possible a mapping between the PPN language and the STEP, AP-214 is necessary. This mapping process will be presented in the next chapter.

chapter v

process specifications in step-ap214

As described in (Adlemo, Andreasson, Fabian, Gullander, Hellgren, Lennartson, Liljenvall and Pernebo 1997) there is a great need for efficient exchange of information from the resources to the routing specifications in order to create truly flexible systems. The main idea in (Adlemo et al. 1997) is that a routing specification holds information about operations to be executed and technical demands on the same operations, while the resources present what they can do. This is to be executed in real time so that a product can be in a certain state, wanting to execute its next operation and asks which resource that is free and able. There is always the risk of a resource braking down in the middle of an operation and should this happen, the product is forced to move on to another resource instead.

The approach in the present work is somewhat different. It is more static in the sense that when a route, or alternative routes, have been created they are fixed throughout the system until changes are made and a new route or routes are calculated. The advantages of this approach is that it uses an international standard for information exchange and is therefore easily applied to many different systems, industries, and companies.

The present thesis aims at making use of the international standard, ISO10303-214, or application protocol 214 (AP214) of the STEP-standard (STandard for Exchange of Product model data) (TC184/SC4 2001), for the communication and storing of process specifications. To begin with the STEP AP-214 was developed to represent product information. In recent years, however, it has been extended to include both process- and resource information as well as product information. The aim of this thesis is therefore to show how process specifications created with the PPN tool can be mapped to the extended STEP AP-214 format. The presented mapping should define the relationship between the information and the DES specification.

Much research has already been conducted, both on information modelling, e.g. (Schenk and Wilson 1994), (Scheller 1990), and (Eversheim, Marczinski and Cremer 1991) as well as on discrete event modelling, e.g. (Cassandras and Lafortune 1999), (Hoare 1985). However, little has been investigated concerning the connection between information and discrete event models, i.e. how an information structure could be mapped to a discrete event structure of a process plan. In this research the aim has been to create a method for the automatic generation of DES specifications according to the STEP standard. There are several reasons for doing this. Companies do not become dependent on one system, but are able to change supplier at any time. The standard also enables companies to have control of all their information and offers the possibility of withdrawing information from the specific systems that are used to develop information into a unified framework based on the STEP standard. The different specifications can then be created using the specific tools that are most suited for a certain task.

5.1 ISO 10303

The purpose of the STEP standard is to create a unified framework for sending and receiving data to and from different systems, companies and so on. It is also an attempt at making it possible to reuse already created information, and in this sense not having to constantly ”reinvent the wheel”.

STEP is an international standard that ”provides a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged” (TC184/SC4 1994). The term *exchange* should be interpreted as the exchange of data between computer systems in environments associated with the complete life-cycle of a product, including manufacturing. Application protocols define the scope, context, and information requirements for a particular application, e.g. the automotive industry (AP214), or the electrical design and installation (AP212). An application protocol is divided into two different representations of the information requirements: the application reference model (ARM) and the application interpreted model (AIM). The ARM is used to capture the information requirements using application-based terminology i.e. terminology that is understood by the domain experts of that particular application. For instance, in AP214 terms like *wheel space* and *overall axle distance* would be used, because they are widely used in the automotive industry. The AIM provides a mechanism for inter-operability between different application protocols to, e.g. describe mechatronic products by using both AP214 and AP212. In-depth information on application protocols and STEP in general are available in (Warthen 1990, Kemmerer 1999, Owen 1993).

5.1.1 EXPRESS

The EXPRESS language is a formally specified and structured language (Schenk and Wilson 1994) used to define the ARM models in STEP. The EXPRESS language is an earlier alternative to the Unified Modelling Language (UML). Usually, the ARM is also defined in EXPRESS and often presented in EXPRESS-G, a graphical subset of the EXPRESS Language.

The basic constructs of EXPRESS (and EXPRESS-G) are the entity and the attribute. An entity is similar to a class in object-oriented programming, i.e. it is a representation of something of interest in the real world. The attribute is a kind of property, and as such it represents a particular aspect of an entity.

In EXPRESS-G an entity is graphically represented as a box with a name in it, cf. Figure 5.1. The name is the identifier of the item it represents in the real world. Attributes are represented by a line ending with a small circle, showing the direction of the relationship. They are labelled with the name of the attribute, as well as any cardinality constraints. A dashed line represents an optional attribute whereas a thick line represents a supertype-subtype relationship, i.e. the same as inheritance in object-oriented programming, using e.g. Unified Modelling Language (UML) (Booch, Rumbaugh and Jacobson 1999), cf. Figure 5.2. A supertype, i.e. the parent of an inheritance relationship, can be abstract (ABS) meaning that the entity can not be populated with data.

The model in Figure 5.1 is conceptual and will be used to present the AP214 standard. Another type of model will also be used to exemplify the use of AP214, the so-called instantiated model, which is a model populated with data from the real world. cf. example in Figure 5.3. The triangle in the lower right corner of the entities in Figure 5.3 indicates that it is an instantiated model. In some instances a filled triangle will occur, indicating that

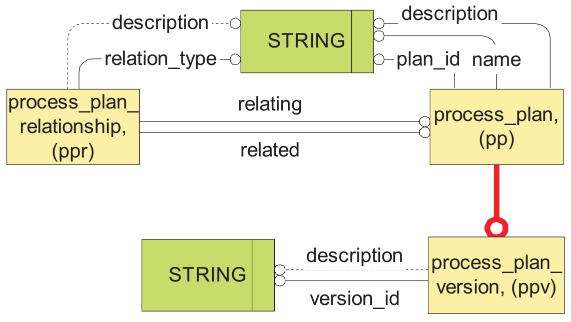


Figure 5.1: Example of entities represented in the EXPRESS language.

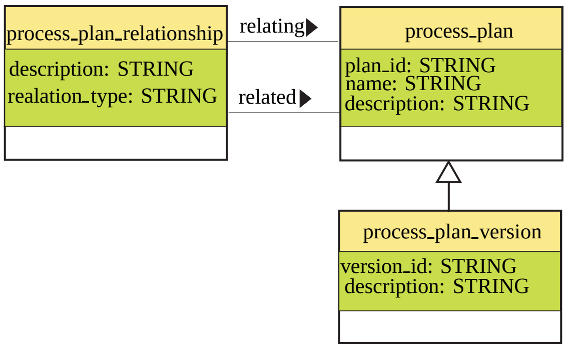


Figure 5.2: The same model as in 5.1 given as an UML model.

all mandatory attributes are instantiated, for example the *process_plan_relationship* entity in Figure 5.3. A transparent triangle on the other hand indicates that some, or all, of the mandatory attributes have been left out.

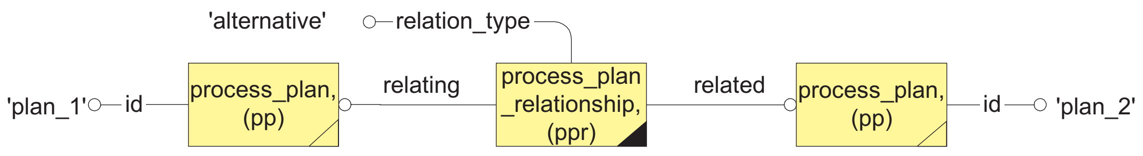


Figure 5.3: Instantiated models of the model in Fig. 5.1. Two process plans relating to each other by an alternative relationship which means that "plan_2" is an alternative process plan to "plan_1".

5.2 Application protocol 214

AP214 is an application protocol developed to meet the information exchange needs of the automotive industry. However, (Johansson 2001b) has shown that the generic structure of AP214 can be used to represent not only cars, but any type of mechanical product, including a manufacturing resource.

Figure 5.4 shows the different parts of the design and production project that STEP AP214 includes. As can be seen in this figure, three different main areas are described: product, process, and resource (manufacturing system). *Control code* and *Behavior models* are not included in the standard and are therefore not marked in gray.

In order to create the *behavior* model and the *control code*, which are not described in STEP, using the information given by the STEP standard, a mapping i necessary. The mapping maps information from a discrete event model into the product, process and resource part of the STEP standard in Figure 5.4.

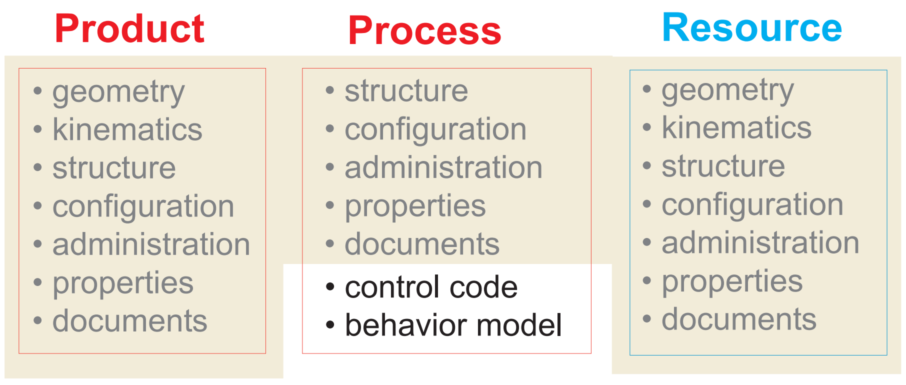


Figure 5.4: Graph describing three different parts in STEP AP214. These parts are product, process and resource. All the information marked in gray is included in the standard. Not included is thus *control code* and *behavior models* in the process part.

5.2.1 Process Model

The process model in AP214, cf. Figure 5.5, is the holder of all necessary process information, such as the process plan identifier, relationships between processes, sequences etc. The process model consists of a structure to hold meta-data about a process plan. This structure is identified by the *process_plan* (pp) in Figure 5.5. A *process_plan* consists of one or more processes represented by the *process_operation_occurrence* (poo). The *process_operation_occurrence* represents the occurrence of a process in a process plan. More specifically, it represents the occurrence of a definition of a process, the *process_operation_definition* (pod). This mechanism enables the reuse of a definition in several different places in the same process plan, as well as in several different process plans.

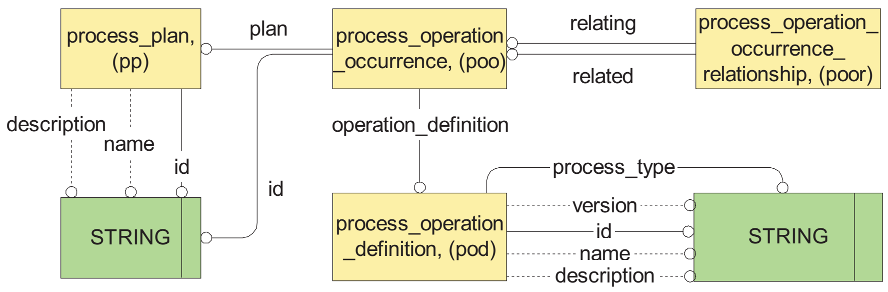


Figure 5.5: Populated process model representing the sequence between two processes.

The relationship between two *process_operation_occurences* is represented by the *process_operation_occurrence_relationship* (poor) where the attribute *relation_type* holds the type of relationship. The attribute *relating* points in the direction of the *process_operation_occurrence* prior to the *process_operation_occurrence* pointed out by the attribute *related*.

5.2.2 Resource Model

The manufacturing resources can be represented in several different ways in AP214, depending on the level of detail and the design life cycle stage. Two different representations have been identified as important, the *single_instance* and the *physical_instance*, cf. Figure 5.5.

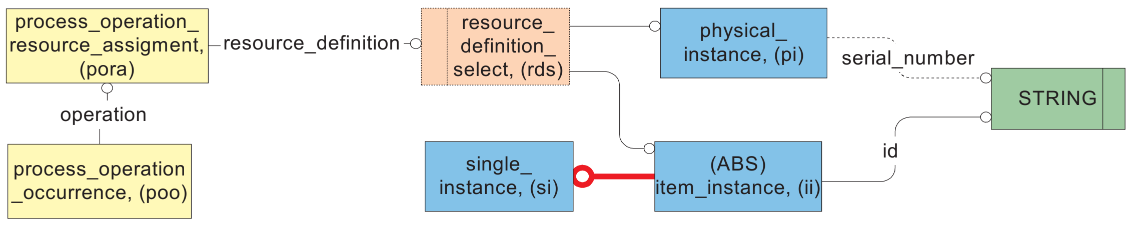

process_operation_ resource_assigment, (pora)
resource_definition
resource_ definition_ select, (rds)
physical_ instance, (pi)
serial_number
STRING
operation
process_operation _occurrence, (poo)
single_ instance, (si)
(ABS) item_instance, (ii)
id

Figure 5.6: Representation of resource data in AP214.

The *single_instance* and the *physical_instance* are both instances of an abstract representation of a manufacturing resource (*item*), but there is one significant difference between them. The *single_instance* represents one occurrence of a certain type of manufacturing resource, whereas the *physical_instance* represents a physical resource on the shop floor. Thus the *single_instance* is better used for planning purposes before a physical resource exists, while the *physical_instance* is better used when a physical resource already exists.

5.2.3 Operation and interlocking Model

The main attribute is *condition_assignment*, which connects the *process_operation_ occurrence* with resource information, i.e. *physical_instance*, via *state_assignment*. The *condition_ assignment* defines a state condition that has to be fulfilled before a *process_operation_ occurrence* can be executed. A *condition_assignment* can refer to a number of *state_ assignments*, which describes that there could be more than one resource state that has to be fulfilled. The *condition_assignment* has an attribute *name*, which can be either ”interlock” or ”required resource state”. The ”required resource state” describes that the condition is for an EOP specification, while ”interlock” naturally specifies that it belongs to an interlock specification.

The entity *state_assignment* refers to *state*, which holds the actual state information. *State_assignment* also refers to *physical_instance* so that the state can be associated with a specific resource.

In this context it is important to note that because application protocol AP214 is used for representing processes, process relations, and resources, but is unable to represent resource states it is necessary to also use AP239. In particular the entities *State*, *State_assignment*, and *Condition_assignment* (von Euler-Chelpin, Holmstrm and Richardsson 2004, Falkman et al. 2004). In this way it is thus necessary to combine two application protocols in order to represent EOP and IL specifications.

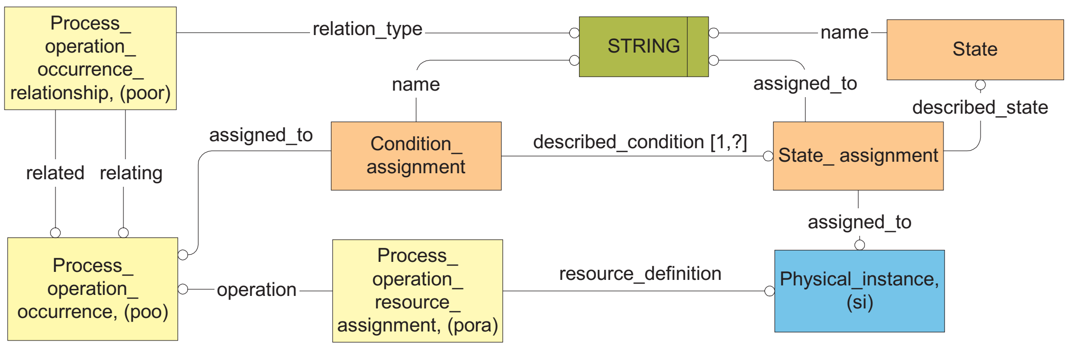


Figure 5.7: Representation of conditions for *process_operation_ocurrence* described as a STEP/Express model.

5.3 Example: PPN specification to STEP/Express model

In this example the connection between the PPN language and the STEP-AP214 standard is shown. This is done by creating STEP/Express models of the same example that was given in Chapter 4 where a high level routing specification was created using the PPN language.

5.3.1 Relations of operations specification

The ROP specification in Fig. 4.10 in Chapter 4 is modelled as a STEP/Express model in Fig. 5.9. Each *poo* refers to a *pod*, which describes the actual operation with the attribute *name*. The required resources for each operation are described by the *pora* relating to *poo*. A *pora* in its turn relates to the specific *pi*, which describes which resource that is required. Please note that two *poo*s relate to the same *pod* and that a *pora* refers to each of the two *poo*s. The relation between the two *poo*s is ”substitution” (alternative in PPN) described by the *poor*. This means that it models an operation O_3, in Fig. 5.9, that can be executed by either using resource R_{3a} or resource R_{3b}.

The arbitrary order execution of operations O_1 and O_2 is modelled by the attribute *relation_type* for *poor_01* equal to ”exclusiveness”. Both operations O_1 and O_2 refer to operation O_3 with the *relation_type* for both *poor_02* and *poor_03* equal to ”sequence”. This specifies that operation O_3 is executed when both operation O_1 and O_2 have finished their respective executions. Both operations O_1 and O_2 also refer to operation O_4 and O_5 with *relation_type* equal to ”sequence”. These relations are however left out for clarity. Operations O_4 and O_5 refer to each other with the *relation_type* for *poor_06* ”simultaneity” specifying that these can be executed in parallel. The *poor_06* between O_3 and O_6 has the *relation_type* ”simultaneity” specifying that these can be executed in parallel. There are also *poor* with *relation_type* ”simultaneity” between O_3 and O_4 as well as O_3 and O_5, these are however also left out here for clarity. In the PPN specification in Fig. 4.10 this parallel behavior is split into two descriptions, both the PN and the process algebra is used. The fact that operation O_6 has to wait until operations O_4, and O_5 have finished their respective executions is specified in Fig. 5.8 using the *relation_type* equal to ”sequence” between, O_4 and O_6, as well as between O_5 and O_6.

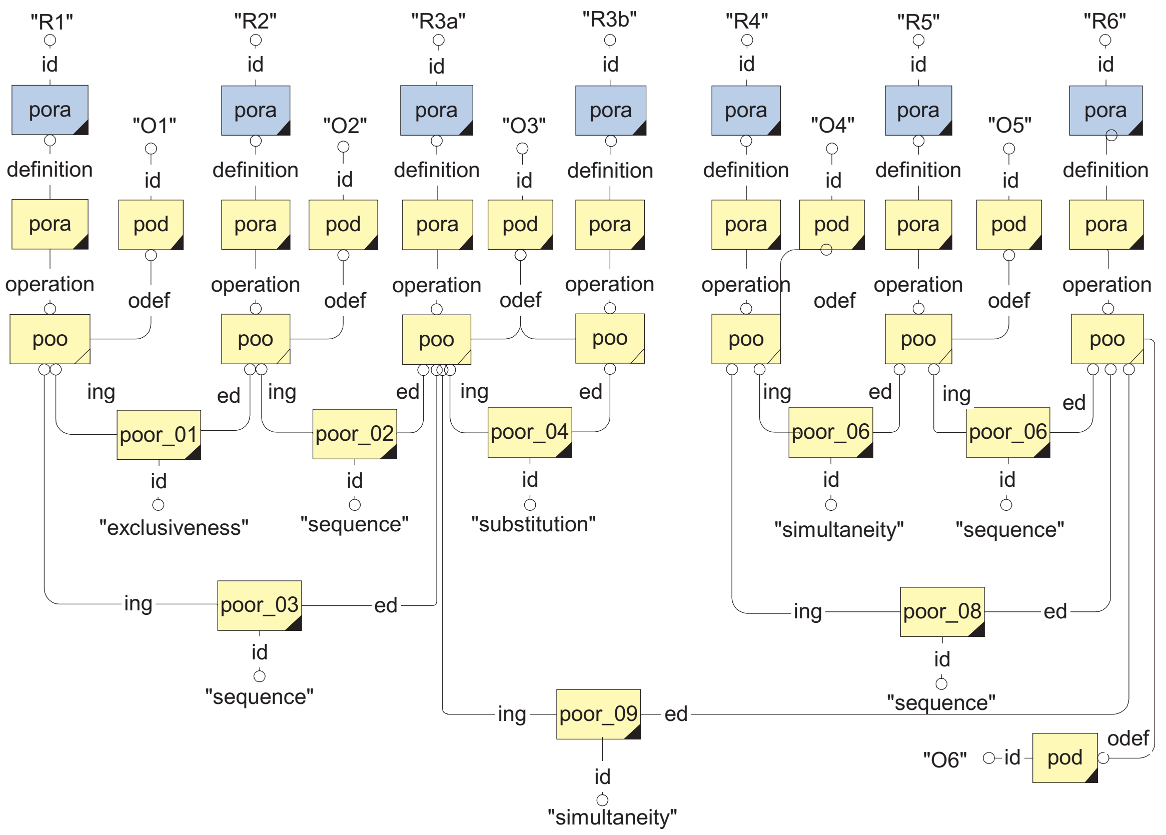


Figure 5.8: The ROP specification in Fig. 4.10 given as an instantiated STEP/Express model.

5.3.2 Execution of operations specification

The same EOP specification as described in Fig. 4.11(a) in Chapter 4 is in Fig. 5.9 described as a STEP/Express model. The two components are described by the *physical_instances* with the attribute *id* equal to the components identity. Every *pi* refers to a *pora*, which in turn refers to a *poo*, as described in Section 5.2.2. A *condition_assignment* refers to the state *state_assignments*, which refers both to the specific resource as well as its state.

The three events in the EOP specification in Fig. 4.11(a) are described by three *pod* entities. Each of these is referred to by a *poo*, which in turn is referred to by a *condition_assigment* with the attribute ”required resource state”. The *condition_assignment* refers to a *state_assignment* that both points out the required state as well as the actual component. The sequence between the events is described using the *poor* entity with the attribute ”sequence”.

5.3.3 Interlocking specification

The interlocking specification in Fig. 4.11(b) in Chapter 4 is specified using a STEP/Express model in Fig. 5.10. In this figure a single restriction for event $goDown$ for component R_{6a} is described using a *condition_assignment* entity. The attribute *name* is in the *condition_assignment* equal to ”interlock” and refers to the *poo*. The required state specified by the *condition_assignment* is described by the *state* via the *state_assignment*.

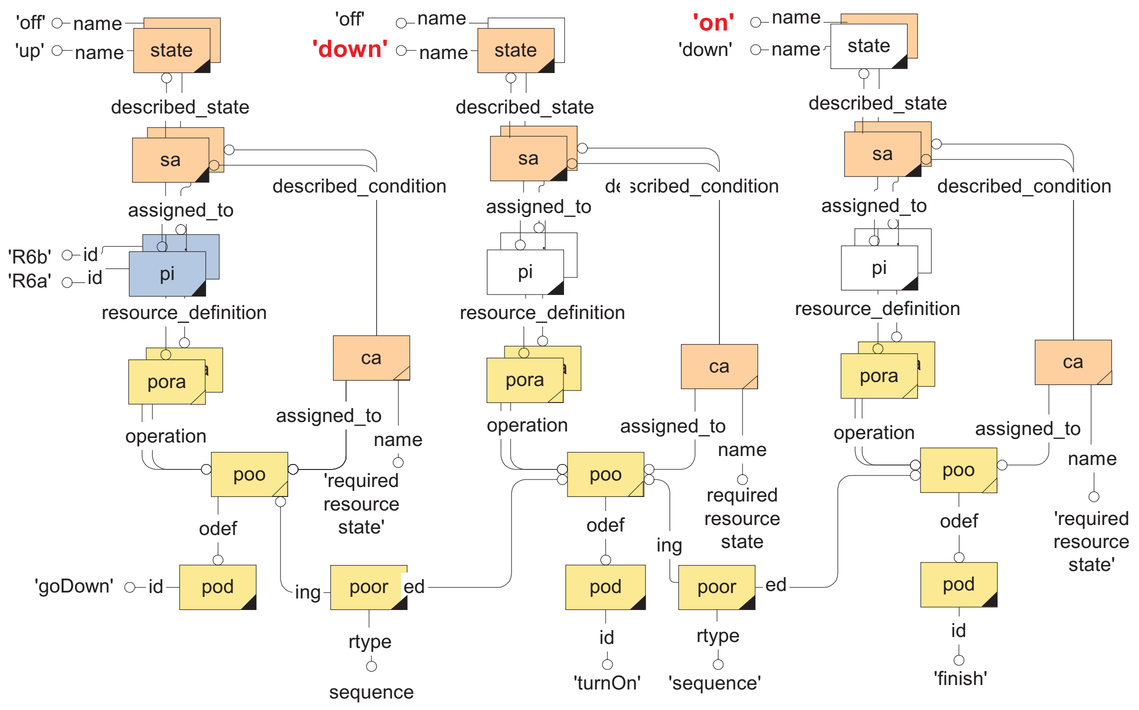


Figure 5.9: EOP specification of operation O_6 in Fig. 4.11(a) given as an instantiated STEP/Express model.

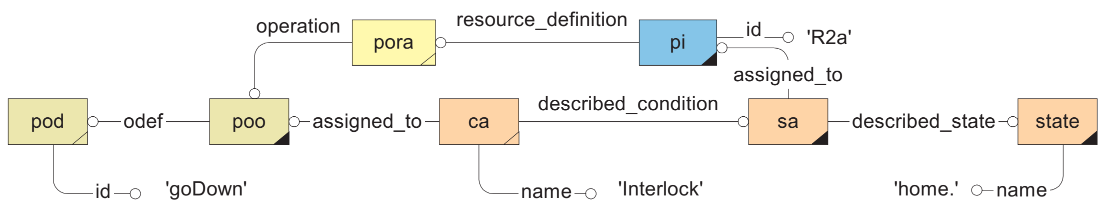


Figure 5.10: Interlocking specification for *goDown* in Fig. 4.11(b) described as a STEP/Express model.

5.4 Discussion

The ability to use the PPN language as a tool for specifying process-related information in a formal way is, as described earlier, of great importance. It is, however, also important that information is only created once and then distributed to other users, e.g. departments within a company. This is important in order to decrease the risk of specifying the same thing more than once and also in avoiding the risk of having more than one correct version of a specification. For this reason a connection between the PPN tool and the STEP standard has been created. As discussed earlier this is beneficial for the efficient exchange of information between different computer systems and users. This enables different users to only extract the information needed for their specific purposes, i.e. a high level routing specification involves different kinds of information and a specific user might, for example, be interested in only knowing which resources that a certain product requires. This chapter has shown how the same high level routing specification given as a PPN model in Chapter 4 can be mapped to a STEP description.

chapter vi

summary of included papers

In this section the papers included in the thesis will be given a brief presentation. There are five included papers appearing in chronological order, reflecting the development of research activities. The papers have been reformatted for uniformity, but are otherwise unchanged.

6.1 Paper I

Falkman, P. and Lennartson, B. and Åkesson, K. and Fabian, M. (2005), A High Level Specification Language based on Process Algebra and Petri Nets, Submitted to: *IEEE Transactions on Automation Science and Engineering*.

A formal high level specification language combining process algebra and Petri nets is presented. The language, called process algebra Petri net (PPN), is a combination of the compact expressions of process algebra and the visual strength of Petri nets. Compared to earlier work in the same field, the PPN language is more focused on high level constructs, readability, and visual strength. The language is created to be used for the specification of flexible production systems, especially resource allocation systems. The specifications created by the PPN language are then used as a base for supervisor synthesis using supervisory control theory (SCT). More specifically, a process algebra is suggested where the process operators express the same process relations as are possible in the international standard ISO-10303-214 or STEP-AP214. STEP-AP214 standardizes the exchange of product, process and resource information, thereby enabling a more efficient information exchange between product and manufacturing system design, also making it beneficial to be able to use the presented tool for creating specifications according to this standard. The operators defined in order to automatically generate specifications according to the STEP-214 standard are sequence, alternative, arbitrary order (in STEP exclusiveness), synchronization and parallel execution. A relation between the suggested PPN language and ordinary labelled Petri nets is also presented.

6.2 Paper II

Falkman, P. and Lennartson, B. and Åkesson, K. (2005), Formal Specification of Flexible Robot Cell using Process Algebra Petri Nets, Submitted to: *IEEE Transaction on Control System Technology.*

In this paper a real industry case is used to show how the PPN language, presented in paper I, can be used for the specification of discrete event systems, more specifically flexible manufacturing systems. The industry case used involves products manufactured in a robot cell. The studied robot cell is located at Volvo Car Corporation, Torslanda, Sweden and is used for producing the V70 and S80 cars. The cars are specified in such a way that parts of the necessary control of the robot cell can be generated automatically, i.e. the resulting specifications is used as a base for supervisor synthesis. The synthesis is performed using a program called *Supremica* implementing the supervisory control theory (SCT) (Ramadge and Wonham 1987), which deals with the interaction between the (controlled) plant and the (computer) supervisor. Finite state automata (FSA) is the modelling language used in *Supremica* and we show in this paper how PPN models are formally converted into finite state automata.

6.3 Paper III

Falkman, P. and Lennartson, B. and Tittus, M. (2005), Specification of a Batch Plant using Process Algebra and Petri Nets, Submitted to: *Control Engineering Practice.*

In paper III the objective is to show how the PPN language, defined in paper I, can be utilized in order to simplify the specification of desired routes of a chemical batch process. By doing this, we point to the generality of the PPN language. It is also made evident how parts of a specification may be more ideally expressed with either Petri nets or process algebra. Based on the created specifications a supervisor may also be synthesized, which synchronizes the utilization of available resources. The language is illustrated for different combinations of multiple and alternative resource allocation systems, especially for a batch-process problem that follows throughout the paper, where process operators such as synchronization and alternative choice are shown to be very powerful. Two additional building blocks are also introduced in the PPN language. These may be used to model material transfer in both flexible manufacturing systems and batch processes. These building blocks are the split and join operations. The join operation combines two or more material flows and the split operation separates two or more material flows. The join and split operations are exemplified using the batch plant example that is developed throughout the paper.

6.4 Paper IV

Falkman, P. and Nielsen, J. and Lennartson, B. (2003), Automatic Generation of Object Models for Process Planning and Control Purposes using an International standard for Information Exchange, In: *Journal of Systemics, Cybernetics and Informatics*, Vol 1, Number 5.

In paper IV a formal mapping between static information models and dynamic models is presented. The information structure is given according to the ISO10303-214 or the STEP-standard (STandard for Exchange of Product model data). The mapping is achieved by analyzing the information structure, (ISO 10303-214), and the dynamic structure, (the MPPN-model) which was introduced in (Falkman and Lennartson 2001) and represents an earlier version of the PPN language introduced in paper I, in order to gain knowledge of the semantics of their respective objects and structures. The gained knowledge is then synthesized to result in the semi-formally defined mapping model. Finally, the result is validated using a case study at Scania Oskarshamn, Sweden. This is done by populating the ISO 10303-214 model with data from the Scania case, and then implementing the mapping method in order to automatically create an MPPN-model based on the Scania data.

6.5 Paper V

Falkman, P. and Nielsen, J. and Lennartson, B. and Euler-Chelpin, A. (2005), Automated Generation of STEP AP214 models from Discrete Event Systems for Process Planning and Control, Submitted to: *IEEE Transactions on Automation Science and Engineering.*

The present research is a more thorough and complete description of the ideas presented in paper IV, and aims at making use of an international standard, ISO10303-214, or application protocol 214 (AP214) of the STEP-standard (STandard for Exchange of Product model data) (TC184/SC4 2001), for the communication and storing of process specifications. The suggested tool for producing process information uses the PPN language presented in Paper I. Paper V shows how process specifications created with the PPN tool can be mapped to the STEP AP-214 format. The presented mapping defines the relationship between the information and the DES specification. The created DES specifications are used for simulation and also for verification and automatic supervisor synthesis. The detailed control, also specified using the PPN language, involves specific control of each resource. The result is validated through a case study at Volvo Car Corporation, Torslanda, Sweden. This is done by creating PPN-specifications based on the Volvo data and then implementing the mapping method in order to automatically generate an ISO 10303-214 model.

chapter vii

concluding remarks

The market needs of today are rapidly changing, making it increasingly important for engineering companies to meet with growing demands on flexibility and ability to decrease their time to market while still maintaining product quality - all at a low cost. One piece of the puzzle in achieving this is to make the information exchange between product design and manufacturing systems design more efficient. A much shortened iteration cycle could be obtained if information about product design solutions could be made instantly available for engineers involved in manufacturing systems design.

The present thesis deals with the specification of discrete event systems, especially resource allocation systems. To the best of our knowledge, despite the many formal specification languages already presented in previous research, efficient tools for producing specifications which are concise as well as easy to read and interpret are still lacking. A common limitation of formal languages presented in previous research is that their strength lies in formal evaluation techniques rather than in visual strength. To be employed as an engineering tool, however, it is important that a formal specification language also results in a clear and easy-to-read description of what is specified. This will increase the understanding of created specifications and thereby also decrease the risk of mistakes happening due to misinterpretations. In this thesis an attempt has therefore been made to present such a tool, based on a formal high level specification language. This language, called process algebra Petri nets (PPN), combines process algebra with ordinary labelled safe Petri nets. The PPN language is an extension, generalization and formalization of the earlier MPPN language (still used in Paper IV). The suggested process algebra defines two new operators, i.e. the synchronization and the arbitrary order operators. These operators have proven to be very useful for the purpose of creating the desired high level routing specifications.

The fact that both Petri net constructs and algebra expressions can be used in order to decrease specification complexity also makes PPN a flexible language. The high level routing specifications include relation of operations (ROP), execution of operation (EOP) as well as interlocking (IL) specifications which are described on different levels of detail. That the PPN language is able to provide descriptions on all these different levels is yet another piece of evidence for it being a flexible language. This is also shown by the fact that the PPN language is able to accommodate restriction expressions. These expressions have proven to be very valuable when it comes to decreasing the complexity of the high level specifications. They have also been a necessity when modelling the EOP and IL specifications where these are specified by changes of resource states.

A method is also presented that formally converts the PPN models into finite state automata, and thus existing formal evaluation techniques for simulation, verification, and supervisor synthesis can be easily applied.

The presented language defines a limited process algebra where the process operators express the process relations avaible in the international standard STEP-AP214. To the best of our knowledge, the PPN language constitutes a first attempt at using a formal language in order to create a tool that can automatically generate specifications according to the STEP standard. The STEP AP-214 was originally developed to represent product information. In recent years, however, it has also been extended to include both process- and resource information as well as product information. Even though there are a lot of software tools available for the generation of both product- and resource information, there is still no tool for producing process information. In the present thesis, however, such a tool is presented. The presented tool makes use of the PPN language for the generation of process-specifications, i.e. the specifications of discrete event systems, especially resource allocation systems. Even though much research has already been conducted, both on information modelling and discrete event systems, little has been investigated concerning the connection between the two. The present work, however, researches this connection. The presented mapping defines the relationship between the information and the DES specification.

Finally, it can be said that the method introduced guarantees that the expected information is delivered fast and without the potential errors induced by manual handling, something which is crucial when short lead times are required. Due to the fast information exchange it also enables simulation, automatic supervisor synthesis and verification to be conducted early in the development chain.

7.1 Further Research

Further work is of course needed, for example a complete implementation of a tool for the PPN language. This tool should have the capacity to create high level specifications according to the PPN language, translations of PPN models into finite state automata representations, as well as the ability to convert a PPN model into a STEP representation. So far only preliminary implementations have been achieved.

Another area where further work is needed is the ability to translate a finite state automata representation back into a PPN representation. This is important since after supervisor synthesis, the resulting supervisor will be described as finite state automata, which can be very difficult to overview. Thus, in order to obtain a clear picture of how the resulting supervisor is constructed it should be converted into a PPN representation. Preliminary results have recently been obtained by Andersson et al. (2005). Another interesting approach would be to investigate how supervisor synthesis could be performed on PPN models without translating these into finite state automata.

Finally, it would be challenging to apply the suggested ideas to their full extent to a real industry case, where the PPN language would be used for the specification. A first attempt of doing this is planned in a coming project.

included papers

process algebra petri net

Submitted to: *IEEE Transactions on Automation Science and Engineering*:

A High Level Specification Language based on Process Algebra and Petri Nets

P. Falkman and B. Lennartson and
K. Åkesson and M. Fabian
[†]Signals and Systems
Chalmers University of Technology

This paper presents a formal high level specification language combining process algebra and Petri nets, to be used for the specification of resource allocation systems, such as certain classes of flexible production systems. This language, process algebra Petri net (PPN), combines the visual strength of Petri nets with the compactness of process algebra expressions. The aim is to use the specifications expressed in the PPN language as a base for supervisor synthesis using supervisory control theory. A process algebra is suggested where the process operators express the same process relations as are possible in the international standard for information exchange STEP-AP214. The suggested PPN language is a powerful tool for specification of discrete event systems in general, and resource allocation systems in particular, delivering concise and easy-to-read specifications of large complex systems. It is flexible in the sense that both Petri net constructs as well as algebra expressions can be used in order to decrease the complexity for the user. A relation between the suggested PPN language and ordinary labelled safe Petri nets is also presented.

1 Introduction

A discrete event system (DES) is a system that, at any time, occupies one out of a finite set of states, and changes state at the occurrence of an event. Specifications of such systems are often expressed in text-based natural-language documents. A major drawback with this is ambiguity, it is possible to interpret the natural-language specification in many ways. This may lead to misunderstandings that are costly and/or time-consuming if discovered at a late stage of a project. Consequently, it is very important for a costumer and a supplier to agree on a specification, written in such a way that misunderstandings are avoided. A way of trying to avoid this kind of problem is to use a formal specification language.

When it comes to discrete event systems, a number of different formal specification languages have been presented. Some of them are algebraically based, some are based on graphical representations, and still others are expressed by temporal logic.

The term ”process algebra” was coined in 1982 by Bergstra & Klop (Bergstra and Klop 1982). A general process algebra includes processes, events, and operators, which are used to build algebraic expressions describing the discrete behavior of a system. The communicating sequential processes (CSP) language by Hoare (Hoare 1985) and the calculus of communicating systems (CCS) language by Milner (Milner 1980) are the major languages based on process algebra. These languages have been further developed in e.g. (Hopcroft and Ullman 1979, Degano et al. 1987, Olderog 1991, Best et al. 1998, Brinksma 1995). An example of a language based on temporal logic is given in (Rescher and Urquhart 1971).

The Petri net (Peterson 1981) introduced by Carl Adam Petri, finite state automata, see e.g. (Hopcroft and Ullman 1979, Kozen 1997), and StateChart, introduced by Harel (Harel et al. 1987) (an extension of finite state automata), have intuitive and precise graphical representations. The main reason for this is the finiteness of the structures; Petri nets can with a finite number of elements even represent an infinite state-space. This is typically not the case with process algebra and temporal logic; only in very special cases are the graphical representations finite, and thus usefully represented graphically. Combinations of the mentioned languages have also been considered. These include for example process algebra and Petri Nets given in e.g. (Best et al. 2001, Mayr 1997, Best et al. 2002, Pena and Cortadella 1996, Bloom et al. 1997, Jmaiel 2000). A common limitation of the mentioned languages and combinations of them is that their strength lies in formal evaluation techniques rather than in visual strength. As an engineering tool it is however important that a formal specification language also results in a clear and easy-to-read description of what is specified. This will increase the understanding of created specifications and thereby also decrease the risk of making mistakes due to misinterpretations.

The present paper is focussed on specification of resource allocation systems, such as certain classes of flexible production systems. To the best of our knowledge, despite the many formal specification languages already presented in previous research, there still is a lack of efficient tools producing the kind of specifications we desire for this kind of systems, i.e. specifications that are concise as well as easy to read and interpret. The aim of this paper is therefore to present such a specification language. This language combines the graphical features of Petri nets with the compact expressions of process algebra. Compared to earlier work, the focus of this language, called process algebra Petri net (PPN), is to a higher extent on high level constructs, readability, and visual strength. A further aim is to use the created specifications for supervisor synthesis applying supervisory control theory (SCT) (Ramadge and Wonham 1987).

More specifically, a process algebra is suggested where the process operators express the same kind of process relations as in the international standard STEP-AP214 (*ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles* 1994), also more formally known as ISO-

10303-214. STEP-AP214 standardizes the exchange of product, process and resource information. It enables an efficient information exchange between product and manufacturing system design, which is crucial in order to decrease lead times (Richardsson 2005). For this reason it is also beneficial to be able to use the presented tool for creating specifications according to this standard. The operators to be defined in order to automatically generate high level specifications according to the STEP-214 standard (Falkman et al. 2004) are sequence, alternative, arbitrary order (in STEP exclusiveness), synchronization and parallel execution.

The suggested PPN language is an extension, generalization and formalization of results presented in (Falkman et al. 2001, Falkman and Lennartson 2001). Using PPN may deliver compact and concise models of large complex systems. It is flexible in the sense that both Petri net constructs as well as algebra expressions can be used in order to generate easy-to-read specifications and in that way reducing the complexity for the user. In (Falkman and Lennartson 2005b) a method is presented that formally converts the PPN models into finite state automata, which means that existing formal evaluation techniques for simulation, verification, and supervisor synthesis are easily applied.

The paper is organized as follows: first a thorough introduction to the suggested PPN language is given in Section 2 through 4. Recursive processes are described in Section 5, and process requirements are introduced in Section 6. Finally a resource allocation example is given in Section 8.

2 Process algebra Petri Net (PPN)

Process algebra is in (*Dictionary of Algorithms and Data Structures, process algebra* 2004) defined as ”an algebraic theory to formalize the notion of concurrent computation, best exemplified in CSP and CCS”. The Petri net formalism is used to describe concurrent transition systems, but contrary to process algebra it uses a graphical representation. Petri nets consist of places, transitions, and arcs connecting the places to the transitions and vice versa, see e.g. (Peterson 1981). The PPN language, suggested in this paper uses a combination process algebra and Petri nets, allowing a process expression at each Petri net transition. The following sections will define the process algebra and Petri nets of the PPN language.

2.1 Process algebra

A process is defined as

$$P \xmapsto{a} P' \tag{1}$$

which means that a process P can execute an event a and then behave as a process P'.

A process P_1 followed by a process P_2 is expressed using the sequence operator $\rightarrow$ as $P_1 \rightarrow P_2$ defined in Section 3.1. This means that P_2 starts immediately when P_1 has successfully finished its execution. In PPN an atomic process is a process that can only execute one event and then successfully finish. Without confusion we allow the same name for both an event and the corresponding atomic process. A similarity concerning atomic processes can be found in (Aceta, Larsen and Ingólfsdóttir 2004). Using the sequence operator the basic process definition (1) is therefore alternatively expressed as

$$P = a \rightarrow P' \tag{2}$$

using the assignment $=$ defining the identifier on the left to be the process on the right where the atomic process a is followed by the process P'. If $P' = b$ then $P = a \rightarrow b$. Throughout the present paper, lower-case letters will denote events and atomic processes as will also words written in lower-case letters, while general processes are denoted with upper-case words or letters.

Processes and states

There is no conceptual difference between processes and their states (Aceta et al. 2004). Take a process $P_1 = a \rightarrow b \rightarrow P_3$, which can be described as $P_1 = a \rightarrow P_2$ and $P_2 = b \rightarrow P_3$. Then P_1, P_2 and P_3 can also be considered as states of the process P_1. In general the name of a process coincide with the name of its initial state.

Furthermore, we will use a process F to specify that a process has successfully finished arriving at the final state F. This process is necessarily introduced in some definitions, but is otherwise not explicitly expressed. The process $P_1 = a$ ends in the state F after executing the event a. This process obviously has two states, see Fig. 1, denoted P_1 (initial state) and F (final state).

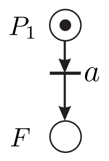


Figure 1: Process $P_1 = a$ given as a PN.

Process alphabet

The alphabet for a process P_i, Σ_{P_i}, is a finite set of events. This can be divided into two disjunct subsets, $\Sigma_{P_i} = \Sigma^e_{P_i} \dot{\bigcup} \Sigma^{ne}_{P_i}$. $\Sigma^e_{P_i}$ is a set of events executed by process P_i and $\Sigma^{ne}_{P_i}$ is a set of events that are never executed.

Process operators

In this paper two sets of operators will be defined, *binary* and *unary* process operators. The *binary* process operators involve two (or more) processes. These operators are sequence, alternative, arbitrary order, synchronization and parallel, denoted as follows

$$\rightarrow, \; +, \; \oplus, \; \&, \; \parallel \tag{3}$$

We will use $*$ to denote a generic binary operator.

The *unary* process operators are operators that only involve one process and these are the start and stop operator, denoted as

$$P^{\uparrow}, \; P^{\downarrow}, \tag{4}$$

respectively.

Note that all operators are defined for processes, which include atomic process expressions such as $a + b$, $a \& b$, and $a \rightarrow P$.

Operational semantics

In order to formally define the behavior of each algebraic operator so-called Structured Operational Semantics (SOS) (Plotkin 1991) will be used. SOS is a method for the precise description of the meaning of operators in labelled transition systems.

The definition of an operational semantics for a language will usually take the form of a set of inference rules, which can be seen as implications, defining the valid transitions of the system. The intention is to infer $P_1 * P_2 \xmapsto{c} P_1' * P_2$ from $P_1 \xmapsto{c} P_1'$, given two processes P_1 and P_2. Process P_1 performs event c and then behaves as process P_1'. In SOS this is expressed in the following way

$$\frac{P_1 \xmapsto{c} P_1'}{P_1 * P_2 \xmapsto{c} P_1' * P_2'}$$

with $P_1 \xmapsto{c} P_1'$ as the hypothesis and $P_1 * P_2 \xmapsto{c} P_1' * P_2'$ as the conclusion. The transition rule may also have a side-condition which is placed to the right of the actual transition rule.

2.2 Petri Nets

The PPN language uses ordinary labelled safe Petri nets (PNs). This means that there is at most one token in each PN place at any time. The PN part of the PPN language may be used to model simple sequences, but can very well be used to model more complex constructs such as parallelism and loops.

PNs used in the PPN language have one *initial* place and one *connector* place. This correspond to a PN that start (and end) with a single place marked, i.e. a single place with a token. This means that the PN models only allow one place with a token in the initial marking vector. This is no restriction since a PN with more than one token in the initial marking vector can always be reconfigured such that only one place has an initial token, see Fig 2. Both the initial and the connector place can have loops, see Section 5. The initial and the connector places are used when converting the PPN models into PN. The connector place of a PN, in the PPN, describes that a single place then has a token, in much the same manner as for the initial place. The reason for these restrictions is to simplify the translation from PPNs to PNs. The extra transitions that are added in order to realize this are uniquely labelled with a start event sta_i and a stop event sto_i.

Alphabet and language

The alphabet for a PN_i, Σ_{PN_i}, is a finite set of events and $\Sigma_{PN_i}^*$ is the set of all finite strings of symbols from Σ_{PN_i} including the empty string ϵ. Since only safe PNs are used in the PPN language these can always be represented by a finite reachability graph with finite set of events $\sigma_i \in \Sigma_{reachPN}$ and with a finite set of states $Q_{reachPN}$. This reachability graph can always be described by a finite state automaton. The language for a reachability

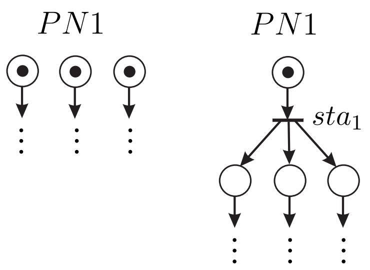


Figure 2: Example of how a PN with several initial tokens in several places is reconfigured such that only one place has an initial token.

graph $L(reachPN_i)$ is therefore the same as for the language of a corresponding finite state automata (Hopcroft et al. 2001), i.e. a regular language.

The modified language $L'(reachPN_i)$ of a reachability graph is defined as

$$L'(reachPN_i) = proj_{\Sigma'}(L(reachPN_i)) \tag{5}$$

using a function $proj$, defined as

$$proj_{\Sigma'}(s\sigma) = \begin{cases} proj(s) \;\; if \;\; \sigma \in \Sigma' \\ proj(s)\sigma \;\; else \end{cases} \tag{6}$$

$$proj_{\Sigma'}(\epsilon) = \epsilon \tag{7}$$

and describes that if the next event after a string s is in Σ' then this event is removed.

2.3 PPN

The PPN language allows a process expression at each PN transition, see e.g. Fig. 4b. There are no definite rules on how to combine PN and process algebra in the PPN language, which means that the PN part can be used to model not only simple sequences but also more complex parts of a specification, see Fig. 12b. The aim is to create specifications of DES using the PPN language. Since a specification specify the allowed language $L(reachPN)$ it is no limitation to assume that only deterministic specifications are considered for the PPN language. Thus, a language based notation of equivalence is sufficient for our purposes.

Equivalence

Two PNs, $\Sigma_{PN_1} = \Sigma_{PN_2}$, are assumed to be equal if they have the same alphabets and their respective reachability graphs have the same language, $L'(reachPN_1) = L'(reachPN_2)$. That is,

$$PN_i \backsimeq PN_j \Rightarrow \begin{cases} \Sigma_{PN_i} = \Sigma_{PN_j} \\ L'(reachPN_i) = L'(reachPN_j) \end{cases} \tag{8}$$

Two PPN models, $PPN_1 \backsimeq PPN_2$, are assumed to be equal if their corresponding PNs are equal $PN_1 \backsimeq PN_2$. That is,

$$PPN_i \backsimeq PPN_j \Rightarrow PN_i \backsimeq PN_j \tag{9}$$

The translation from PPN to PNs is defined later on. As previously mentioned two sets of operators, *binary* and *unary*, will be introduced in the present paper, and in the following sections these operators are defined and exemplified. Two simple processes $P_1 = a$ and $P_2 = b \rightarrow c$ will be used throughout the paper to exemplify different operators. Corresponding PNs are shown in Fig. 3.

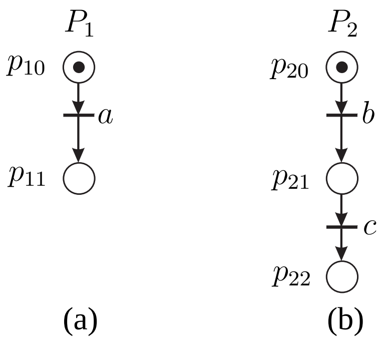


Figure 3: Processes $P_1 = a$ and $P_2 = b \rightarrow c$ given as PNs.

3 Binary process operators

The *binary* process operators to be defined in this section are sequence $\rightarrow$, alternative $+$, arbitrary order $\oplus$, synchronization $\&$ and parallel $||$. The alphabet for a new process $P_1 * P_2$ is $\Sigma_{P_1 * P_2} = \Sigma_{P_1} \bigcup \Sigma_{P_2}$.

3.1 Processes in sequence

As was noted in Section 2.1 the notation

$$P_1 \rightarrow P_2 \qquad (P_1 \;\; then \;\; P_2) \tag{10}$$

expresses a process which behaves first like a process P_1 and then as a process P_2. Process P_2 will begin its execution as soon as process P_1 has finished its execution, in other words when it has reached its connector state.

A PN representation of the process $P_1 \rightarrow P_2$ is obtained by first creating a PN representation of each process involved, and then merge the connector place in P_1 and the initial place in P_2 into one single place. The initial place for the new process $P_1 \rightarrow P_2$ is given by the initial place of P_1 and the connector state is given by the connector place of P_2. This is similar to concatenation in (Peterson 1981).

This is illustrated in Fig. 4 where the two processes are used to model the sequence $P_1 \rightarrow P_2$ of Fig. 3. The transitions in a PPN can involve more than one process as in Fig. 4a or only one process at each transition as in Fig. 4b. The same sequence is in Fig. 4c modelled as a PN with explicit events at each transition. Note that the PN place p_1 in Fig. 4a-4c is the same, which also is the case for place p_2.

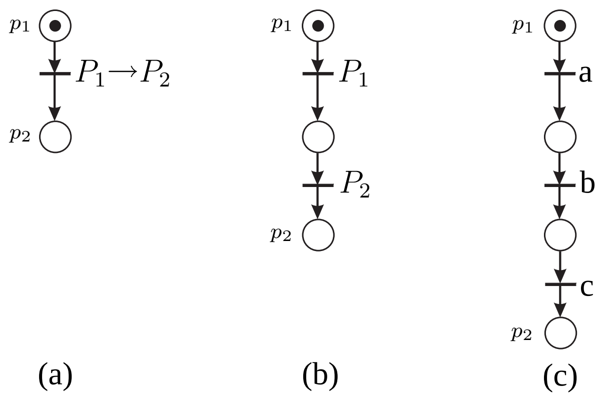


Figure 4: The process $P = P_1 \rightarrow P_2$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

The laws governing the sequence operator are as follows

L1:	$F \rightarrow P_1 \backsimeq P_1$
L2:	$P_1 \rightarrow F \backsimeq P_1$
Associative:	$P_1 \rightarrow (P_2 \rightarrow P_3) \backsimeq (P_1 \rightarrow P_2) \rightarrow P_3$

3.2 Choice between alternative processes

The expression

$$P_1 + P_2 \qquad (P_1 \ \ or \ \ P_2) \tag{11}$$

denotes a process, which behaves either like a process P_1 or as a process P_2 but not both.

The transition rules for the alternative (choice) operator are

$$\frac{P_1 \xmapsto{a_1} P_1'}{P_1 + P_2 \xmapsto{a_1} P_1'} \tag{12}$$

$$\frac{P_2 \xmapsto{a_2} P_2'}{P_1 + P_2 \xmapsto{a_2} P_2'} \tag{13}$$

The first of these two rules defines that if process P_1 can perform event a_1 and subsequently behave like P_1', then process $P_1 + P_2$ is capable of the same transition. The second alternative rule (13) is symmetric to the first one (12). In the continuation of this paper, when rules are symmetric, only the first of the rules will be described due to limited space.

The expression $P_1 + P_2$ can be represented as a PN by first creating a PN representation of each involved process, and then merging the initial place of P_1 and the initial place of P_2. The connector places of each PN are also merged. This is no restriction since (11) behaves like either of the processes. As is described in Section 5, there are restrictions on how P_1 and P_2 can be specified when composing them with the alternative operator.

The alternative choice between the two processes in Fig. 3 is modelled in Fig. 5 both as PPNs and PN. Observe that the PN places p_1 and p_2, respectively, in Fig. 5a-5c are the same.

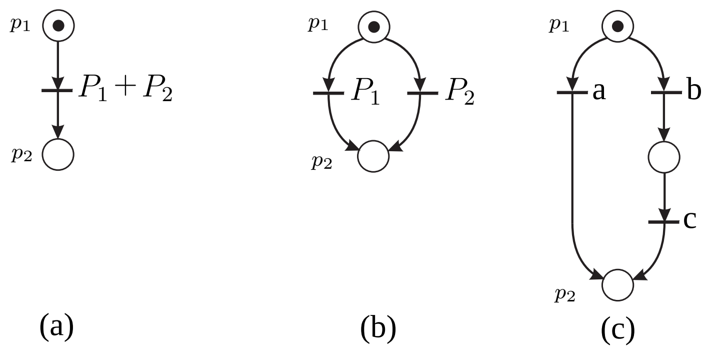


Figure 5: The process $P = P_1 + P_2$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

The algebraic laws governing the alternative operator are as follows

L1:	$P_1 + F \backsimeq P_1$
Idempotence:	$P_1 + P_1 \backsimeq P_1$
Symmetry:	$P_1 + P_2 \backsimeq P_2 + P_1$
Associative:	$P_1 + (P_2 + P_3) \backsimeq (P_1 + P_2) + P_3$

Note that $\rightarrow$ has higher precedence than $+$. The alternative operator is not distributive with respect to the alternative operator, i.e. $P_1 \rightarrow (P_2 + P_3) \not\backsimeq P_1 \rightarrow P_2 + P_1 \rightarrow P_3$. This is due to the fact the choice in the expression $P_1 \rightarrow (P_2 + P_3)$ is made after process P_1 has finished its execution, while in the expression $P_1 \rightarrow P_2 + P_1 \rightarrow P_3$ there is a non-deterministic choice.

3.3 Processes in arbitrary order

An arbitrary order operator between events was introduced in (Lennartson, Fabian, Tittus and Hellgren 1998) and is here extended to include processes. This operator can be described by a set of processes that are all to be executed, but not at the same time. This means that they must be executed in a sequence, and if the order does not matter this will imply that there are $n!$ correct sequences where n is the number of processes.

There are two main reasons for introducing this operator. The first, and maybe most obvious, is that it provides a compact way of expressing alternative sequences e.g. $P_1 \rightarrow P_2 + P_2 \rightarrow P_1$. The second and equally important reason is that this operator is defined in the international standard STEP (TC184/SC4 1994). In order to automatically generate specifications according to STEP it is advantageous to define the same operator in the PPN language.

The arbitrary order operator can generally be expressed as

$$P = \oplus\{P_i, \ldots, P_n\} = P_{perm}(A) \tag{14}$$

where A is the set of involved processes and the recursive function P_{perm} is defined as

$$\begin{aligned} P_{perm}(\Omega) &= \{\textstyle\sum_{P_i \in \Omega}(P_i \rightarrow P_{perm}(\Omega - \{P_i\}))\} \\ P_{perm}(\emptyset) &= F \end{aligned}$$

where Ω is a set of involved processes. This recursive function returns all permutations as a deterministic process algebra expression if none of the involved processes have the same first event.

The arbitrary order between the two processes in Fig. 3 is demonstrated in Fig. 6.

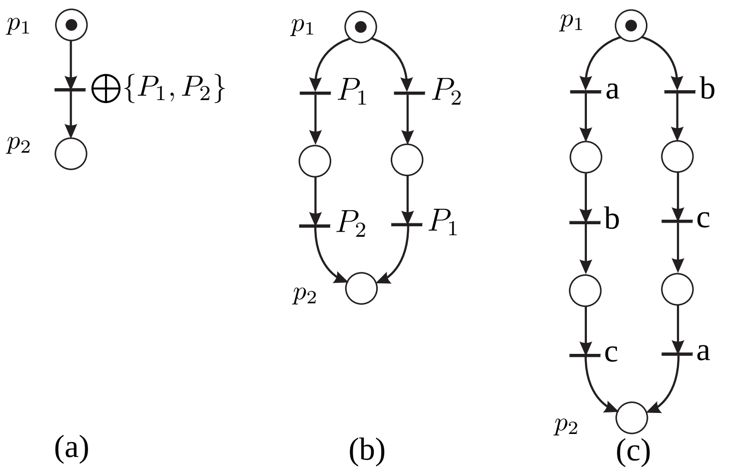


Figure 6: The process $P = \bigoplus\{P_1, P_2\}$ is in (a) and (b) given as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

3.4 Explicit process synchronization

There are many synchronization operators defined in the literature, e.g. composition in CCS $|$ (Milner 1980), full synchronous composition (FSC) in CSP $\|$ (Hoare 1985), general parallelism in LOTOS $P_1|[a_1, \ldots, a_n]|P_2$ (Bolognesi and Brinksma 1987), and parallelism $\|$ (Arnold 1994). However, for flexibility and reuseability we do not consider those operations appropriate. Instead we propose the explicit synchronization operator $\&$, which is a modification of the synchronization operator between events in (Lennartson et al. 1998). A synchronized process $P_1 \,\&\, P_2$ specifies that both the first event of each involved process need to execute simulatively as well as last event of each process. If a synchronized process only involves an atomic process, then this will be synchronized with the first event of the other synchronized processes, and then taking no part in the synchronization of the last events.

The explicit synchronization operator $\&$ is useful when flexibility and reuseability is desired. Consider the following example: the task is to synchronize the processes P_1 and

P_2 at one moment in time and later on P_1 with another process P_3. To specify these synchronizations by FSC it is necessary to introduce specific event labels for these two different situations in the corresponding models. By the suggested synchronization operator $\&$ a specification process S including $P_1 \& P_2$, later followed by $P_1 \& P_3$, does not require the processes to be modified. This is very useful when modelling resource allocation systems, as shown in Section 8. A similar concept for event synchronization was suggested by Fabian (Fabian and Lennartson 1994), but then denoted coupled events. Similar ideas for event synchronization can also be found in (Arnold 1994, David and Alla 1992, Stirling 1996). Note that the event synchronization (avoiding relabelling) is generalized in this paper to process synchronization.

The explicit synchronization between two processes P_1 and P_2 is denoted

$$P_1 \& P_2 \qquad (P_1 \ synchronized \ with \ P_2) \tag{15}$$

Process P_1 has a set Σ_{s_1} of start events. There can be a single event in Σ_{s_1} or, if process P_1 begins with an alternative, more than one event. Process P_1 also has a set Σ_{f_1} of final events. Two similar sets, Σ_{s_2} and Σ_{f_2}, are in the same way defined for process P_2. The transition rules for this synchronization operator use these sets in order to define the intended behavior. A PN representation of the expression $P_1 \& P_2$ include two parallel paths and therefore two extra transitions modelling the parallelism, see Fig. 7(b), in order to achieve an initial and a connector place. These two transitions are uniquely labelled with a start event sta_i and a stop event sto_i, respectively. Processes involved in an synchronization can cannot be recursive processes, see Section 5.

The first and most obvious rule (16) describes that both processes are able to execute either their start events or final events. The process $P_1 \& P_2$ then executes the synchronized event $a_1 \& a_2$ and after that behaves as $P_1' \& P_2'$.

$$\frac{P_1 \xmapsto{a_1} P_1' \quad P_2 \xmapsto{a_2} P_2'}{P_1 \& P_2 \xmapsto{a_1 \& a_2} P_1' \& P_2'} \ \Big\backslash \ \begin{array}{l} a_1 \in \Sigma_{s_1}, a_2 \in \Sigma_{s_2} \ \text{or} \\ a_1 \in \Sigma_{f_1}, a_2 \in \Sigma_{f_2} \end{array} \tag{16}$$

Note that if $a_1 \in \Sigma_{f1}$ and $a_2 \in \Sigma_{f2}$ then $P_1' \& P_2' = F$. Rule (17) defines that if a_1 is the next event in P_1 and a_1 is neither the start or final event, i.e. $a_1 \notin \Sigma_{s_1} \cup \Sigma_{f_1}$, then P_1 can execute a_1 independently of P_2. This parallel behavior is described by executing event a_1 and subsequently behaving as process $P_1' \& P_2$ in

$$\frac{P_1 \xmapsto{a_1} P_1'}{P_1 \& P_2 \xmapsto{a_1} P_1' \& P_2} \ \Big\backslash \ a_1 \notin \Sigma_{s_1} \cup \Sigma_{f_1} \tag{17}$$

A symmetric rule for a_2 is not presented.

The last transition rule, (18), defines that if one process P_2 is successfully finished F, then the other process P_1 will execute its remaining events independently, resulting in $P_1' \& F$.

$$\frac{P_1 \xmapsto{a_1} P_1' \quad F}{P_1 \& F \xmapsto{a_1} P_1' \& F} \tag{18}$$

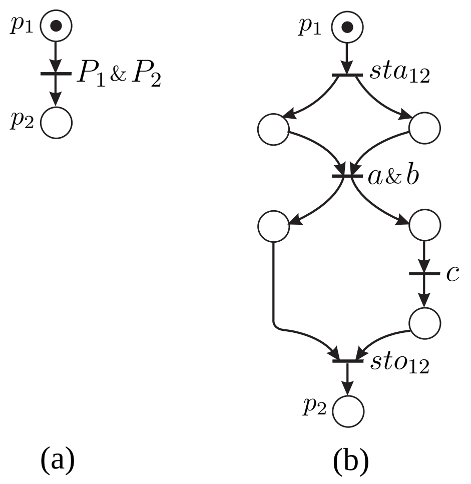


Figure 7: The process $P = P_1 \& P_2$ given in (a) and (b) as PPN models using the processes $P_1 = a$ and $P_2 = b \rightarrow c$.

The explicit synchronization of the two processes described in Fig. 3 is demonstrated in Fig. 7, where we note that the transition rule in (18) is applied.

Example 7 – Synchronization

The synchronized process $P_3 \& P_4$ is given as a PPN specification in Fig. 8a. The two processes $P_3 = a \rightarrow b$ and $P_4 = c \rightarrow d \rightarrow e$ are synchronized by their first and last events respectively, resulting in the specification seen in Fig. 8b. This means that two synchronized events are created, $a\& c$ and $b\& e$. Events executed in a process after the first event, and before the final event, are executed independently of the other processes. In our example this means that event d in P_4 can execute without any concern for process P_3.

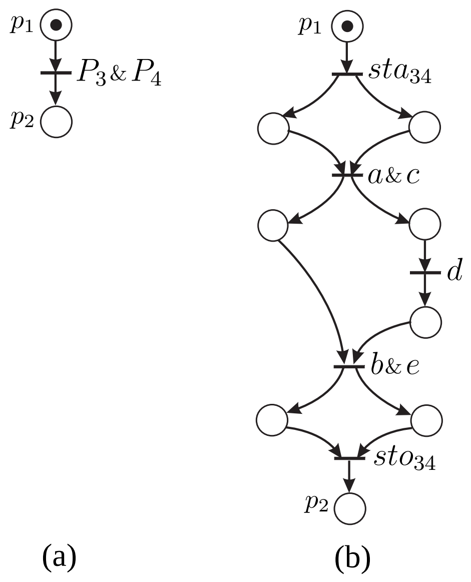


Figure 8: The process $P = P_3 \& P_4$ given in (a) and (b) as PPN models using the processes $P_3 = a \rightarrow b$ and $P_4 = c \rightarrow d \rightarrow e$.

□

The algebraic laws governing the explicit synchronization operator are

L1:	$P_1 \,\&\, F \backsimeq P_1$
Symmetry:	$P_1 \,\&\, P_2 \backsimeq P_2 \,\&\, P_1$
Associativity:	$P_1 \,\&\, (P_2 \,\&\, P_3) \backsimeq (P_1 \,\&\, P_2) \,\&\, P_3$
Distributive:	$P_1 \,\&\, (P_2 + P_3) \backsimeq (P_1 \,\&\, P_2) + (P_1 \,\&\, P_3)$

The synchronization operator $\&$ has higher precedence than $+$ and $\rightarrow$. Note that the sequence operator is not distributive with respect to the synchronization operator, i.e. $P_1 \,\&\, (P_2 \rightarrow P_3) \not\backsimeq P_1 \,\&\, P_2 \rightarrow P_1 \,\&\, P_3$.

3.5 Parallel processes

A parallel execution offers the possibility for processes to execute independently of each other. The expression

$$P = \sqcup\{P_1, \ldots, P_n\} \qquad (P_i \;\; in \;\; parallel) \tag{19}$$

denotes a process P which executes all processes P_i in parallel. If the alphabets of all processes involved are disjunct they will execute independently, see Fig. 9, resulting in interleaving, cf. (Hoare 1985). If, on the other hand, processes have common events these events are synchronized, see Example 9 (event e). The operator $\sqcup$ is an extension of Hoares full synchronous composition (FSC) (Hoare 1985), involving the explicit process synchronization operator $\&$. This parallel operator is not associative and it is therefore required that all involved processes are synchronized at the same time and in a given order. When converting the expression $\sqcup\{P_1, \ldots, P_n\}$ to a PN, parallel paths, one for each process involved, are created and two extra uniquely labelled transitions are created, as in the synchronous case, with a start and stop event. The start transition connect to the initial place of each involved process and the connector places of each process connects to the stop transition.

In the transition rule (20) for the parallel operator given, consider the index set $A = \{1, \ldots, n\}$ covers all involved processes, cf. (19), and the index set $B = \{\ell_1, \ldots, \ell_m\}$ which includes the processes taking part in the synchronized transition in (20); hence $B \subseteq A$. Each process P_{ℓ_i} represented in B defines a set E_i of all events involved in the synchronized event for the specific process. For each process transition there exists at least one event common to more than one other process when $m > 1$. This means that for all elements ℓ_i in the set B there is at least one element $\ell_j \in B$ for which $E_i \cap E_j \neq \emptyset$. It is also possible for the number of involved processes in B to be one, i.e. $m = 1$, which specifies a single process performing an event independently of the other processes (interleaving).

The rule in (20) defines that m processes are executed in parallel. Each of these processes specifies a transition involving a single event, or a synchronized event if $|E_i| > 1$. A new synchronized event is created for the parallel process $\sqcup\{P_1, \ldots, P_n\}$, which is a synchronization of all involved single and synchronized events.

$$\frac{P_{\ell_1} \xmapsto{\underset{i \epsilon E_1}{\&} \sigma_i} P'_{\ell_1}, \ldots, P_{\ell_m} \xmapsto{\underset{i \epsilon E_m}{\&} \sigma_i} P'_{\ell_m}}{\sqcup\{P_1, \ldots, P_n\} \xmapsto{\underset{i \epsilon E_1}{\&} \sigma_i \,\&\, \ldots \,\&\, \underset{i \epsilon E_m}{\&} \sigma_i} \sqcup\{P'_1, \ldots, P'_n\}} \tag{20}$$

with $P_i' = P_i$ if $i \notin B$. Note that $\sigma_i \& \sigma_i = \sigma_i$ and $\sigma_i \& \sigma_j = \sigma_j \& \sigma_i$.

Example 8 – Processes with disjunct alphabets The parallel execution of the two processes in Fig. 3 is illustrated in Fig. 9. The alphabets of these two processes are disjunct $\Sigma_{P_1} \cap \Sigma_{P_2} = \emptyset$, which means that P_1 and P_2 can execute in parallel independently of each other. Observe that the PPN model in Fig. 9b is only valid if the alphabets of the involved processes are disjunct. Example 9 illustrates both independency and synchronization.

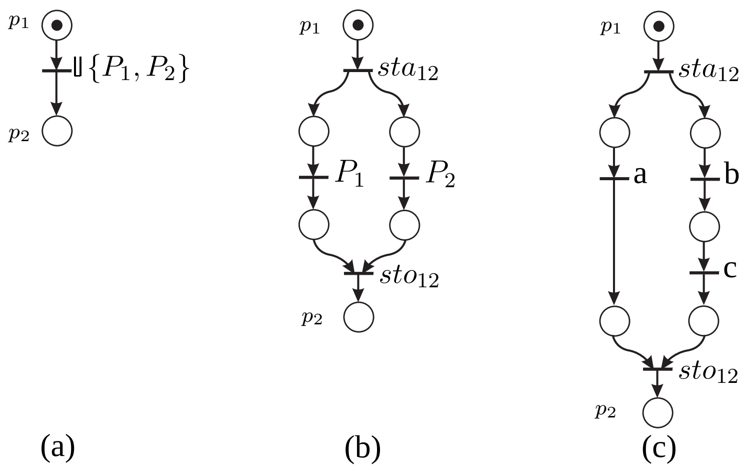


Figure 9: The process $P = \Cup\{P_1, P_2\}$ given as PPN models in (a) and (b). In (c) it is given as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

□

Example 9 – Processes with common events Two processes are given as $P_3 = d{\rightarrow}e\&f{\rightarrow}g$ and $P_4 = h{\rightarrow}e{\rightarrow}m$ with the common event e. These are to be executed in parallel (as shown in Fig. 10). The synchronized event $e\&f$ in process P_3 is required to occur at the same time as the event e in P_4.

□

Example 10 – Processes with common events Three processes $P_5 = a_1\&a_2$, $P_6 = a_2\&a_3$, and $P_7 = a_3\&a_4$ are to be executed in parallel. Process P_5 has event a_2 in common with process P_6 and processes P_6 and P_7 have event a_3 in common. Executing these three processes in parallel results in the synchronized event $a_1\&a_2\&a_3\&a_4$, ie. $\Cup\{P_5, P_6, P_7\} = a_1\&a_2\&a_3\&a_4$.
□

Relabelling

Since the focus of the suggested language is on the specification of complex systems, one major aim has been to keep the final specifications as compact, concise and readable as possible. It is therefore also desirable that a relabelling operator can be left out. The introduced parallel operator $\Cup$ is however unfortunately not associative. As a consequence only monolithic verification and synthesis can then be performed. Recently, modular verification

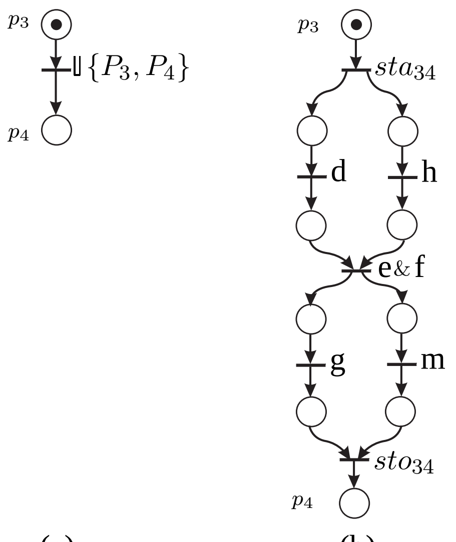


Figure 10: The process $P = \Cup\{P_3, P_4\}$, with $B = \{3, 4\}$, given in (a) as a PPN model and in (b) as a PN (including synchronized events). The processes P_3 and P_4 are defined in Example 9.

and synthesis has become a powerful way of dealing with state explosion for large complex systems (Åkesson, Flordal and Fabian 2002a). For this reason it is advantageous to use an existing parallel operator that is associative e.g. Hoares full synchronous composition (FSC). In order to use FSC a relabelling is required though. This relabelling, see Appendix, is performed when the final specifications are to be used for supervisor synthesis and does not influence the PPN language.

4 Unary process operators

In this section we will introduce a few additional operators, which operate on a single process. The alphabet for a new process $P^{\uparrow}$, $\Sigma_{P^{\uparrow}}$, is the union of $\Sigma^{ne}_{P_i}$ and the first executed event of process P_i and in the same way $\Sigma_{P^{\downarrow}}$, is the union of $\Sigma^{ne}_{P_i}$ and the last executed event of process P_i.

4.1 Start and stop operators

Start and stop operators denote the starting and ending of the execution of process P. The start operator declares that a process will start by executing its first event. It is therefore used to control when a process is allowed to start. In the same way the stop operator tells that a process will finish by performing its last event. Consequently it can be used to control when a process is permitted to finish.

In Fig. 11 the sequence $P_1 \rightarrow P_2^{\uparrow}$ is illustrated. This expression says that P_1 executes before P_2 can start. In this case the continuation of process P_2 is not specified, only the fact that it has to wait until P_1 has finished before it is allowed to start.

Note that the start and stop operators can also be used on process expressions. For instance $(P_1 + P_2)^{\uparrow} \backsimeq P_1^{\uparrow} + P_2^{\uparrow}$ specifies that the first event of P_1 or P_2 is to be executed.

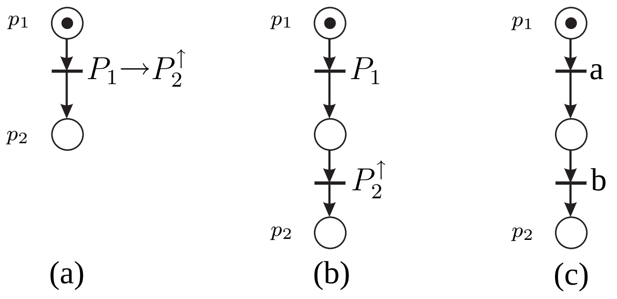


Figure 11: The process $P = P_1 \rightarrow P_2^{\uparrow}$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

Example 11 – Start and stop of processes In this extended example four processes Q_1, Q_2, Q_3 and Q_4 are defined as $Q_i = a_1 \rightarrow b_i \rightarrow c_i$ for $i \in \{1, 2, 3, 4\}$. A specification involving a sequence of three processes is modelled in Fig. 12. The first process is $\Cup\{Q_1^{\uparrow}, Q_2^{\uparrow}, Q_4^{\uparrow}\}$ where A={1,2,4} which implies that Q_1, Q_2 and Q_4 can be started in arbitrary order (executed in parallel). The second process $Q_3^{\uparrow} \& Q_1^{\downarrow}$ implies that the start of Q_3 is to be synchronized with the ending of Q_1. The third and last process $Q_2^{\downarrow} \& Q_3^{\downarrow} \& Q_4^{\downarrow}$ defines that the finishing of Q_2, Q_3, and Q_4 has to be synchronized. The extra transitions and labels created in the conversation to the PPN in Fig. 12(c) and the PN in Fig. 12(c) have been left out for clarity.

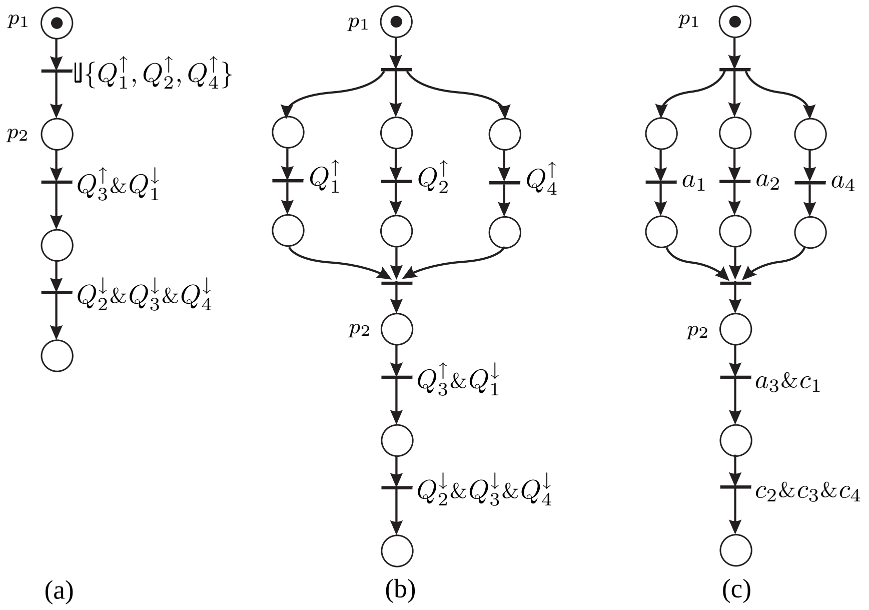

p_1
$\Cup\{Q_1^{\uparrow}, Q_2^{\uparrow}, Q_4^{\uparrow}\}$
p_2
$Q_3^{\uparrow} \& Q_1^{\downarrow}$
$Q_2^{\downarrow} \& Q_3^{\downarrow} \& Q_4^{\downarrow}$
(a)
p_1
$Q_1^{\uparrow}$
$Q_2^{\uparrow}$
$Q_4^{\uparrow}$
p_2
$Q_3^{\uparrow} \& Q_1^{\downarrow}$
$Q_2^{\downarrow} \& Q_3^{\downarrow} \& Q_4^{\downarrow}$
(b)
p_1
a_1
a_2
a_4
p_2
$a_3 \& c_1$
$c_2 \& c_3 \& c_4$
(c)

Figure 12: The specification in Example 11 given in (a) and (b) as PPN models and in (c) as a PN.

□

5 Recursive processes

Recursive behavior can be specified as $P = P_1 \rightarrow P$, where P is repeated after the execution of process P_1. We distinguish between two types of recursive processes, explicit and implicit recursive processes. An explicit recursive process is a process that is present on both sides of the assignment operator $=$, i.e. $P = a \rightarrow P$. An implicit recursive process is a process that involve recursion, i.e. $P = a \rightarrow P_1$ where $P_1 = b \rightarrow P_1$. The initial and the connector states in an explicit recursive process are the same. For a implicit recursive process, $P = a \rightarrow P_1$ where $P_1 = b \rightarrow P_1$ P, is the initial state and P_1 is the connector state.

The stop operator can not be used on explicit nor implicit recursive processes due to the fact that there is no final event. However, when the recursive process is a subprocess as in Fig. 13 the stop operator may still be relevant. There is no limitation for the use of the start operator on recursive processes. The expression $P^{\uparrow}$ simply states that the first event of process P is to be executed.

When specifying an alternative between two processes, $P_1 + P_2$, it is required that none of these processes are explicit recursive processes. This is because the expression $P_1 + P_2$ specifies that only one of the two processes can execute. If there is an alternative between $P_1 = a \rightarrow P_1$ and $P_2 = b \rightarrow c$ the initial states of both processes are merged when converting the expression to a PN. This leads to a choice event a and event b and if event a is executed then the loop takes the process back to the same choice again. This behavior is not what we want to specify when using the $+$ operator.

In Fig. 13 there is a model describing a sequence including three processes $P_1 = a \rightarrow P_1'$, $P_2 = b \rightarrow P_2$, and $P_3 = c \rightarrow P_3'$. Each process P_1, P_2, and P_3 is described as a PN in Fig. 13(a)-(c). The initial place for process P_1 is q_0 and the connector place is q_1. Process P_3 has initial place p_0 and connector place p_1. The initial and connector place for process P_2 is the same r_0. The three Petri nets in a sequence, see Fig 13(d), are put together into an ordinary PN by joining the connector place place q_1 of P_1 and the initial place r_0 of P_2 and the connector place r_0 of P_2 with the initial place p_0 of P_3 in Fig. 13(e). Note that $P^{\uparrow} = a$ and $P^{\downarrow} = c$.

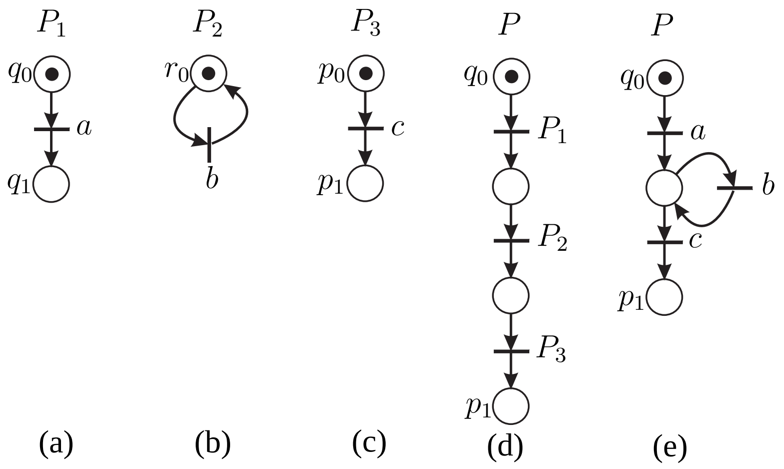


Figure 13: Processes P_1, P_2, and P_3 given as PNs in (a)-(c). Process P given as a PPN in (d) and as a PN in (e).

A recursive process can also involve some alternative processes, which can be written as

$$P = (\sum_{i \in A} P_i) \rightarrow P \tag{21}$$

where there is a choice between a number of sub-processes P_i before the process P is repeated. Observe that the explicit introduction of the sub-processes P_i makes it possible to apply the generic start and stop operators on these individual processes P_i.

Process P, in (21), can be interpreted as a general booking model for a resource which can be booked by a number of tasks (products). Assume that $P_i = a_i \rightarrow b_i$, where a_i represents the booking of resource P by product i and event b_i represents the corresponding unbooking event.

Instead of initially elaborating with a general model and then moving on to a specific model when the actual tasks are given, we suggest that the general form (21) is introduced but with the option to avoid specifying the set A. This kind of model is called a *parameterized model*, since the index i is a free, nonevaluated parameter. Using object-oriented terminology the parameterized model can be considered as a class, while the evaluated model for a given set A is an instance of the model.

This technique is suitable when general and reusable models are required. Parameterized models and events were preliminary introduced in (Lennartson et al. 1998). The concept is related to CCSs value passing calculus, (Milner 1989).

6 Restrictions

The introduction of additional requirements on the execution of processes is highly important when attempting to model discrete event systems in a compact, but yet readable manner. The requirements are described by logical expressions typically specifying when a particular process can begin and/or end its execution.

The logical expressions are introduced inside square brackets, [], following the process to be restricted. The logical expressions specify a specific state (e.g. an initial state) or a set of states. Common logic operators, $\wedge$ (and), $\vee$ (or), and $\overline{P}$ (negation) are allowed inside these expressions.

The expression

$$P = P_1[P_2^{\downarrow} \wedge P_3^{\downarrow}] \tag{22}$$

specifies that a process P_1 is not allowed to start before process P_2 and process P_3 have finished their executions. In other words, P_2 and P_3 must be in their final state before P_1 can begin. Table I.1 summarizes the basic atomic restrictions and related state(s).

The last restriction is typical when a process must wait until a specific event has occurred in another process. Also, observe that $[\overline{P}]$ refers to all states except the initial one, and $[\overline{P^a}]$ refers to all states in P before the event a occurs for the first time.

Finally, note that in a recursive process such as the resource model in (21), the initial state $[P]$ rather represents a free state. This means that $[\overline{P}]$ corresponds to all states except the free state, which means that the resource is occupied.

Table I.1: Atomic restrictions and related state(s).

Restriction	State(s)
$[P]$	Initial state in process P.
$[P^{\downarrow}]$	Final state of process P.
$[P^{a}]$	All states in process P after the event a has occurred (for the first time if it is repeated).

7 Process functions

In this section we will introduce process functions that can be used to further simplify expressions.

In the following example a choice between two processes is made depending on complementary restrictions.

$$A_J(X,Y,Z) = X[Z] + Y[\overline{Z}] \tag{23}$$

The three variables X, Y and Z are processes, and the choice between process X or process Y is determined by the state of process Z, more specifically whether it is in its initial state or not. This function is very helpful when specifying asynchronous join and split that are more thoroughly presented in (Falkman, Lennartson and Tittus 2005).

8 Example: Resource Allocation System

An example cell is illustrated in Fig. 14, where product PT is produced. This cell consists of five robots R_1-R_5, a magazine M_1, two output buffers B_1 and B_2, a fixture F_1, and a conveyor C_1.

Each resource in the robot cell is modelled as a recursive process as described in Section 5, with R_ℓ representing any of the resources, see Fig. 15. Each resource can be booked by a number of routing specifications $1, \ldots, n$.

A single product type PT is manipulated by the cell. The product is a partly assembled car-door. The task of the present cell can be divided into two parts. One is to weld a number of remaining weld spots on the door and also to apply a waterproof paste. The other task is to weld a new part to the existing door. During a work cycle a new door is transported into the cell by the conveyor. The new part is placed in the fixture, and when both door and fixture are in place a number of weld spots is done. Then the manipulated door is moved to one of the two buffers.

Nine different processes $P_1 - P_9$ can be identified for product PT

P_1 : Performs additional welding on parts of the door that has already been assembled. Uses robot R_1. This process may not execute at the same time as process P_2.

P_2 : Applies waterproof paste. Requires robot R_2. This process may not execute at the same time as process P_1.

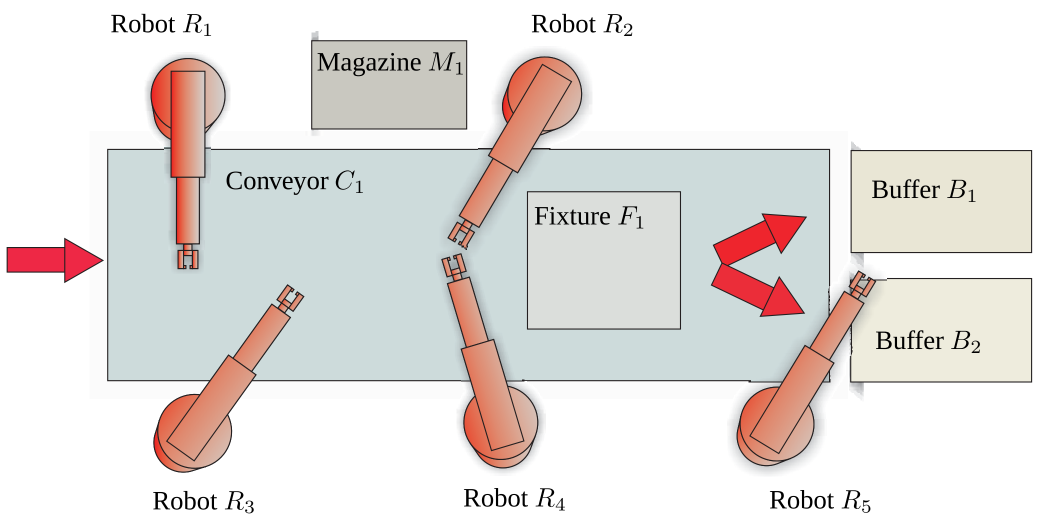


Figure 14: The example cell with five robots $R_1 - R_5$, a magazine M_1, two output buffers B_1 and B_2, a fixture F_1, and a conveyor C_1.

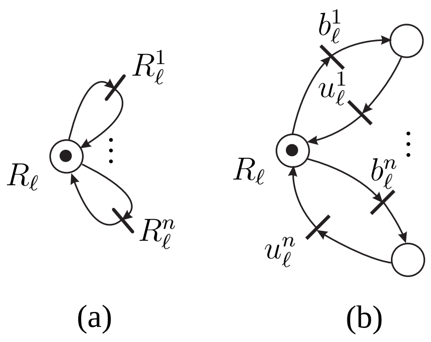


Figure 15: Resource model R_ℓ given as (a) a PPN model and (b) a PN. The resource can be booked by a number of routing specifications $1, \ldots, n$.

P_3 : Places a new part from magazine M_1 into the fixture F_1. Requires resources R_3, F_1, and M_1.

P_4 : Geometrical welding is performed on the new part. Requires resources R_5 and F_1. Has to wait until process P_9 and P_3 have finished their execution.

P_5 : Welds a number of welding spots on the new part using robot R_3. Can not start its execution before process P_4 is done.

P_6 : Welds a number of welding spots on the new part using robot R_4. Can not start its execution before process P_4 is done.

P_7 : Moves the door into one of two buffers, either B_1 or B_2, using robot R_5. All other processes have to have finished their respective execution.

P_8 : A new door is moved into the cell using conveyor C_1.

P_9 : Locks door and fixture to the right position. This process can not execute before P_3.

A specification for product PT is given in Fig. 16 using the PPN language. It is divided into three parallel paths using PN constructs. The path to the left in Fig. 16 describes that processes P_1 and P_2 can execute in arbitrary order by booking and unbooking their respective resource. The right parallel path begins by executing process P_8, i.e. transport a new door to the cell using the conveyor. This is followed by process P_9 that fixates the door and the fixture. This, however, cannot be done until process P_3 is executed, which results in a restriction on P_9. The middle path starts with placing a new part into the fixture in process P_3. This is followed by a geometrical welding specified by P_4. In order for this, process P_9 has to have finished. Then, processes P_5 and P_6 executing in parallel weld the new part to the door. When all parallel paths are finished process P_7 is executed, moving the door using either buffer B_1 or B_2.

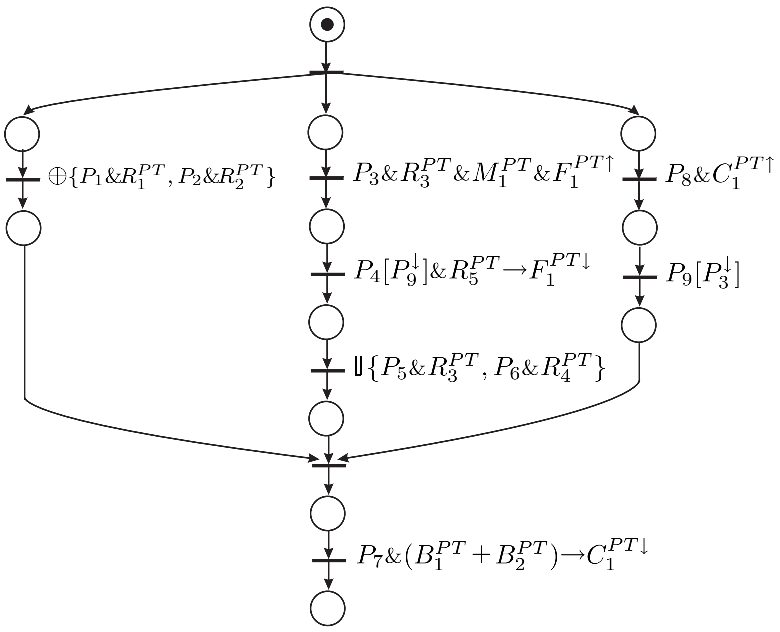

$\oplus\{P_1\&R_1^{PT}, P_2\&R_2^{PT}\}$
$P_3\&R_3^{PT}\&M_1^{PT}\&F_1^{PT\uparrow}$
$P_8\&C_1^{PT\uparrow}$
$P_4[P_9^{\downarrow}]\&R_5^{PT}\rightarrow F_1^{PT\downarrow}$
$P_9[P_3^{\downarrow}]$
$\Cup\{P_5\&R_3^{PT}, P_6\&R_4^{PT}\}$
$P_7\&(B_1^{PT}+B_2^{PT})\rightarrow C_1^{PT\downarrow}$

Figure 16: Specification PT, given as a PPN model specifies the manufacturing of the first product type in the robot cell in Fig. 14.

9 Conclusion and future work

The PPN language presented in this paper defines an algebra where the process operators express the same process relations as are possible in the international standard STEP-AP214 (*ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles* 1994). These include sequence, alternative (choice), parallel, synchronization, and arbitrary order as well as unary process operators for start and stop. PPN also includes the concept of process restriction. The PPN language, to the best of our knowledge, constitutes a first attempt to create a tool, based on a formal language, that can automatically generate specifications according to the STEP standard.

The defined process algebra is combined with ordinary labelled safe Petri nets in order to realize a specification language for discrete event systems, and especially resource allocation systems. It has also been shown how this language can be used for concise and easy-to-read specifications of complex systems, something which is crucial in avoiding costly mistakes. The possibility to translate the PPN specifications into ordinary safe PNs has also been shown. The PPN language has been applied to both batch plants (Falkman et al. 2005) and robot cells (Falkman and Lennartson 2005b), which also shows the generality of the suggested specification language.

An algorithm that performs a translation from a PPN description to a finite state automata representation is presented in (Falkman and Lennartson 2005b). This enables the use of existing formal methods for supervisor synthesis applying supervisory control theory. A method, as well as an algorithm, for the automatic generation of specifications according to the STEP-AP214 from PPN is also being developed (Falkman et al. 2004). Furthermore, a case study of a more complex robot cell at Volvo Car Corporation, Torslanda, Sweden is currently in progress.

Finally it can be said that the suggested process algebra Petri net (PPN) is a powerful language for specifying resource allocation systems, delivering both concise and easy-to-read specifications of large complex systems. The fact that both Petri net constructs and algebra expressions can be used in order to decrease specification complexity also makes PPN a flexible language.

10 Appendix

The relabelling of three PPNs with two transitions each is described in Fig.17.

A transition set (24) for each PN involved in the parallel synchronization is created. An additional transition is added to each transition set. This is called a virtual transition vt and this does not have any events associated to it. The virtual transitions are used in order to represent a transition set which enables transitions not involving all PNs.

$$\begin{aligned} T_{PPN_1} &= \{t_{11}, t_{12}, vt_1\} \\ T_{PPN_2} &= \{t_{21}, t_{22}, vt_2\} \\ T_{PPN_3} &= \{t_{31}, t_{32}, vt_3\} \end{aligned} \tag{24}$$

The cartesian product of all transition sets is

$$T = T_{PPN_1} \times T_{PPN_2} \times T_{PPN_3} = \{\{t_{11}, t_{21}, t_{31}\}, \{t_{11}, t_{21}, t_{32}\}, \ldots, \{vt_1, vt_2, vt_3\}\}$$

All transitions in T are labelled

$$\begin{aligned} \{t_{11}, t_{21}, vt_3\} &\equiv f_1 \\ \{t_{12}, t_{21}, vt_3\} &\equiv f_2 \\ &\vdots \\ \{vt_1, vt_2, vt_3\} &\equiv f_{27} \end{aligned} \tag{25}$$

The set T involves a large number of transitions that are neither relevant nor enabled. It is therefore necessary to exclude these from the set T. Finding out which transitions in T to exclude, it is necessary to know how the involved transitions are connected to each other.

A transition f in T is enabled if its involved transitions are strongly connected components (Cormen, Leiserson, Rivest and Stein 2001) and if no events in f are disabled. In Fig.

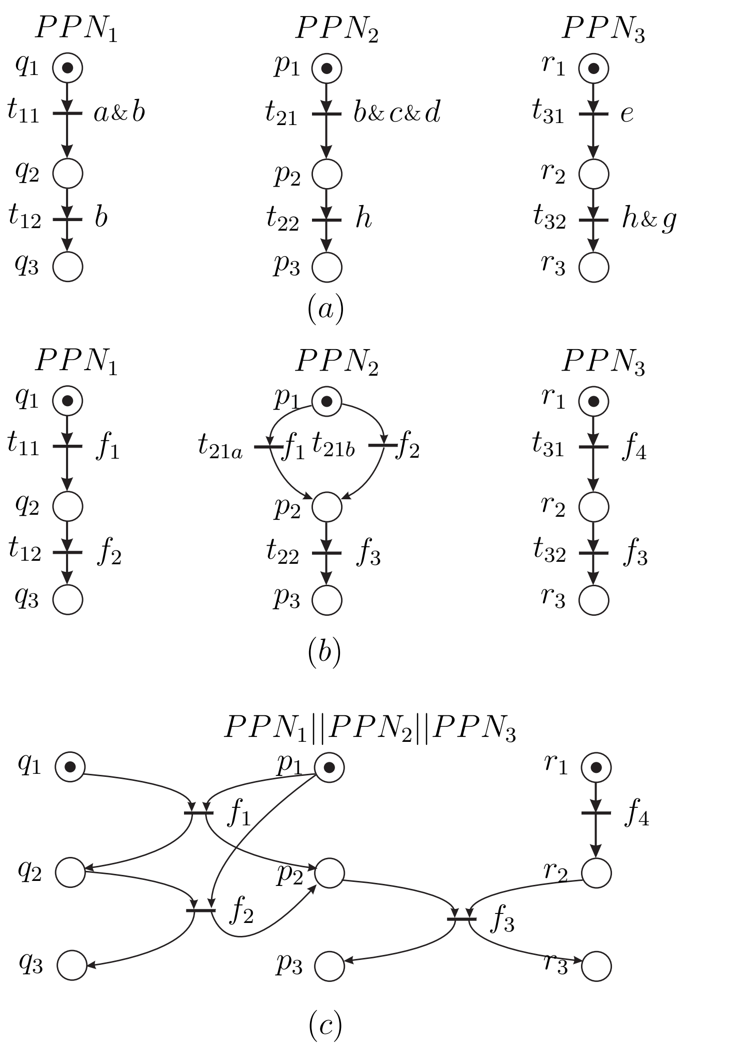


Figure 17: Three PPNs are given in (a) and these are relabelled in (c) and synchronized using Hoares FSC in (c).

17(a) for instance transition t_{11} in PPN_1 enables events a and b and transition t_{21} in PPN_2 enables events b, c and d. Then transition $\{t_{11}, t_{12}, vt_3\}$ is strongly connected component since they connect through event b. If transition t_{31} in PPN_3 would involve event c, transition t_{11}, t_{12}, t_{13} would be strongly connected component since t_{11} connects to t_{12} via event b and t_{13} connects to t_{12} via event c.

A predicate $enab(f)$ can either be true which means that transition f is enabled, or it can be false. If it is false we know that f is not enabled and can be discarded.

$$enab(f) = \begin{cases} true & \begin{array}{l} \text{i} \;\; f\ is\ a\ strongly\ connected\ component \\ \text{ii} \;\; no\ event\ in\ f\ is\ disabled \end{array} \\ false\ otherwise & \end{cases}$$

Algorithm 1 describes how this relabelling is performed. A function $trans(f)$ returns all original PPN transitions t and is used in order to identify which transitions to be relabelled. New PNs are created using $CreateAndAddNewTransition$ where $^\bullet t$ and $t^\bullet$ is the pre and post place respectively.

Algorithm 1: `Relabels all transitions involved in a parallel execution.`

```
Input  : PPN
Output : Relabelled PPN
foreach f such that $enab(f) == true$ do
    foreach $t \in trans(f)$ do
        CreateAndAddNewTransition($^{\bullet}t$, $t^{\bullet}$, $f$)
```

In Fig.17b four new labels are created, f_1, f_2, f_3, and f_4. A synchronization is in Fig.17c performed using Houre's FSC that synchronizes the three PNs with respect to the common there common events.

References

Aceta, L., Larsen, K. and Ingólfsdóttir, A. (2004). An introduction to milners ccs.

Åkesson, K., Flordal, H. and Fabian, M. (2002). Exploiting modularity for synthesis and verification of supervisors, *Proc. of 15'th IFAC World Congress on Automatic Control*, Barcelona, Spain.

Arnold, A. (1994). *Finite Transition Systems: Semantics of Communicating Systems*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Bergstra, J. and Klop, J. (1982). Strong normalization and perpetual reductions in the lambda calculus, *Elektronische Informationsverabeitung und Kybernetik* **18**: 403417.

Best, E., Devillers, R. and Koutny, M. (1998). Petri nets, process algebras and concurent programming languages, *Proc of ICM'98*, Berlin, Germany.

Best, E., Devillers, R. and Koutny, M. (2001). *Petri net algebra*, EATCS monographs on theoretical computer science, Springer, Berlin.

Best, E., Devillers, R. and Koutny, M. (2002). The box algebra = petri nets + process expressions, *Information and Computation* (178): 44–100.

Bloom, B., Cheng, A. and Dsouza, A. (1997). Using a protean language to enchance expressiveness in specification, *IEEE Transactions on Software Engineering* **23**(4): 224–234.

Bolognesi, T. and Brinksma, E. (1987). Introduction to the iso specification language lotos, *Computer Networks and ISDN Systems* **14**(1): 25–59.

Brinksma, E. (1995). Performance and formal design: a process algebraic perspective, *Proc. of Sixth International Workshop on Petri Nets and Performance Models*, IEEE, Durham, NC USA, pp. 124 – 125.

Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2001). *Introduction to Algorithms, Second Edition*, 2nd edn, The MIT Press.

David, R. and Alla, H. (1992). *Petri Nets and Grafcet*, Prentice Hall International (UK) Ltd, Hertfordshire HP2 4RG.

Degano, P., DeNicola, R. and Montanari, U. (1987). Ccs is an (augmented) contact-free c/e system, *in* E. M. Venturini Zilli (ed.), *Mathematical Models for the semantics of Parallelism*, Vol. Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 144–165.

Dictionary of Algorithms and Data Structures, process algebra (2004).
URL: *http://www.nist.gov/dads/HTML/processalgbr.html*

Fabian, M. and Lennartson, B. (1994). Petri nets and control synthesis; an object oriented approach., *Proc of the 2nd IFAC/IFIP/IFORS Workshop on Intelligent Manufacturing Systems, IMS '94*, Vienna, Austria.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P. and Lennartson, B. (2005). Using a high level language for verification and control synthesis of discrete event systems, *Submitted to Transaction on Control System Technology* .

Falkman, P., Lennartson, B. and Tittus, M. (2001). Modeling and specification of discrete event systems using combined process algebra, *Proc. of 2001 IEEE/ASME Advanced Intelligent Mecatronics*, COMO, Italy.

Falkman, P., Lennartson, B. and Tittus, M. (2005). Specification of a batch plant using process algebra and petri nets, *To be submitted to Transactions on Control Engineering Practice* .

Falkman, P., Nielsen, J. and Lennartson, B. (2004). A method for automated generation of discrete event systems from step ap214 for process planning and control, *Submitted to Journal of Manufacturing Systems* .

Harel, D., Pnueli, A., Schmidt, J. and Sherman, R. (1987). On the formal semantics of statecharts., *Proc. of Symposium on Logic in Computer Science.*, pp. 55–64.

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Hopcroft, J., Motwani, R. and Ullman, J. (2001). *Introduction to Automata Theory, Languages and Computation*, 2nd ed. edn, Addison-Wesley Series in Computer Science, Addison-Wesley.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Series in Computer Science, Addison-Wesley.

ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles (1994). ISO standard.

Jmaiel, M. (2000). A unified algebraic framework for specifying communication protocols, *Proc. of Third international Conference on Formal Engineering Methods*, York UK, pp. 57–65.

Kozen, D. (1997). *Automata and Computability*, ISBN 0-387-94907-0, Springer-verlag New York, inc.

Lennartson, B., Fabian, M., Tittus, M. and Hellgren, A. (1998). Modeling primitives for supervisory control, *Proc of WODES '98*, Cagliari, Italy.

Mayr, R. (1997). Combining petri nets and pa-processes, *Theoretical Aspects of Computer Software (TACS'97), volume 1281 of Lecture Notes in Computer Science*, Sendai, Japan.

Milner, R. (1980). *A Calculus of Communicating Systems*, Vol. Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg New York.

Milner, R. (1989). *Communication and Concurrency*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Olderog, E.-R. (1991). *Nets, Terms and Formulas*, Cambridge University Press, Trumpington Street, Cambridge CB2 1RP, Great Britain.

Pena, M. and Cortadella, J. (1996). Combining process algebras and petri nets for the specification and systethis of asynchronious circuits, *Proc of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Fucushima, Japan.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc.

Plotkin, G. D. (1991). *A Structural Approach to Operational Semantics*, Computer Schience Department Aarhus University, DAIMI FN-19, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes, *SIAM J. Control Optim.* **25**(1): 206–230.

Rescher, N. and Urquhart, A. (1971). Temporal logic, *Springer-Verlag, New York* .

Richardsson, J. (2005). *Development and Verification of Control Systems for Flexible Automation*, Licentiate thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 015.

Stirling, C. (1996). *Logics for Concurrency: Structure versus automata*, Springer Verlag, chapter Modal and temporal logics for processes., pp. pp 149–237.

TC184/SC4, I. (1994). Iso 10303-1: Industrial automation systems and integration - product data representation - and exchange - part 1: Overview and fundamental principles, ISO Standard.

converting ppn to finite state automata

Submitted to: *IEEE Transaction on Control System Technology*:

Formal Specification of Flexible Robot Cell using Process Algebra Petri Nets

P. Falkman and B. Lennartson and K. Åkesson
Signals and Systems
Chalmers University of Technology

In this paper a high level specification language, which is a combination of Petri nets and process algebra, is used for specification of discrete event systems, more specifically flexible manufacturing systems. This high level language, called process algebra Petri net (PPN), has been specifically developed in accordance with the international standard STEP-AP214. The PPN language takes advantage of the Petri nets graphical qualities together with the compactness of process algebra, making it possible to create specifications that are both concise and unambiguous. This paper has two aims. The first is to show how the PPN language can be used for specification of a real industry case. The second is to use the resulting specification as a base for supervisor synthesis. This synthesis, based on supervisory control theory (SCT), is performed using a program called *Supremica*, a powerful tool which is able to perform vast calculations on large systems. The modelling language used in *Supremica* is finite state automata (FSA), and this paper therefore shows how PPN models can be formally converted into such FSA.

1 Introduction

The focus of the present paper is on specification of discrete event systems (DESs) (Cassandras and Lafortune 1999). This can be done in a number of different ways, including straight forward textual documents. It is, however, extremely difficult to produce a text-based document that is completely unambiguous. In fact, it is often possible to interpret the textual specification in many different ways, depending on the reader. The result is often misunderstandings, which might be very costly, and/or time consuming, if they are detected at later stages of a project. With this in mind it is natural to recommend formal languages to obtain unambiguous specifications, such that misinterpretations are avoided.

Previous research within the area of DESs has resulted in a number of suggested formal specification languages. These languages are not finished products, rather they are continuously under development to suit new applications. Some are based on process algebra, the communicating sequential processes (CSP) language by Hoare(Hoare 1985) and the calculus of communicating systems (CCS) language by Milner (Milner 1980) being the major examples. These languages have also been further developed in e.g. (Hopcroft and Ullman 1979, Degano et al. 1987, Olderog 1991, Best et al. 1998, Brinksma 1995). Another commonly used, algebraically based, specification language worth mentioning here is Temporal logic (Rescher and Urquhart 1971).

Among graphical languages one of the major examples is the Petri net (Peterson 1981), introduced by Carl Adam Petri in the early 1960s. Another common graphical language is finite state automata, see e.g. (Hopcroft and Ullman 1979, Kozen 1997), where StateChart introduced by Harel (Harel et al. 1987) is an interesting extension including hierarchical structure.

Combinations of the mentioned algebraic and graphical languages have also been considered. These include for example the mixture of process algebra and Petri Nets given in e.g. (Best et al. 2001, Mayr 1997, Best et al. 2002, Pena and Cortadella 1996, Bloom et al. 1997, Jmaiel 2000).

Most of the suggested specification languages, as well as the different combinations of them, have one limitation in common. Their main focus is on formal evaluation techniques rather than on visual strength. In order to really make the best use of a formal specification language, from a user perspective, it is however important that it also results in a clear and easy-to-read description of what is specified. This will not only increase the understanding of the modelled system, but also decrease the risk of unnecessary mistakes being made due to different readers making personal interpretations of certain specifications.

An attempt to create such a language has been made using a combination of Petri nets and process algebra previously presented in (Falkman and Lennartson 2005a). Taking advantage of the Petri nets graphical qualities, together with the compactness of process algebra, this results in a language that can deliver both concise and unambiguous specifications. The suggested language, called process algebra Petri net (PPN), is created in agrement with the international standard for information exchange STEP-AP214 (*ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles* 1994). More details are given in (Falkman, Nielsen and Lennartson 2003a), where a mapping between the STEP standard and the suggested PPN language is presented. One aim of the present paper is to show how this language can be used for specification of a nontrivial DES. This is achieved by specifying a real industry application, a robot cell in a body-in-white production line.

A further aim is to use the resulting specification as a base for supervisor synthesis. This synthesis is based on the supervisory control theory (SCT) (Ramadge and Wonham 1987), which deals with the interaction between a (controlled) plant and its supervisor (computer). The synthesis is performed using a program called *Supremica* (Åkesson et al. 2003). Finite state automata (FSA) is the modelling language used in *Supremica*, and we will therefore in this paper show how PPN models are formally converted into finite state automata.

The industry case used as an example involves products manufactured in a robot cell. The studied robot cell is located at Volvo Car Corporation, Torslanda, Sweden and is used for producing the V70 and S80 cars. The aim is to specify desired operations such that parts of the necessary control of the robot cell can be generated automatically. This supervisor synthesis puts great demands on how both product operations and resources are modelled. In order to maintain a relevant overview of the involved operations, the specification is divided into an hierarchical structure.

The paper is outlined as follows; it starts with an introduction to the suggested high level PPN language in Section 2, including a description on how the specifications are stored and communicated in Section 2.5. Section 3 introduces briefly supervisory control theory and related finite state automata. A description of the translation to finite state automata from process algebra Petri nets is given in Section 4. Finally a larger example is provided in Section 5.

2 Process algebra Petri net (PPN)

A powerful specification language, called process algebra Petri net (PPN), is presented in this section. It makes use of the graphical representations of ordinary labelled safe Petri nets (PNs) as well as the compact representations of process algebra, in order to generate concise specifications of complex systems. A more formal definition of the PPN formalism is given in (Falkman and Lennartson 2005a), where operational semantics are used to describe the behavior of each operator. Operator laws together with the relationship between the PPN models and PNs are also given. Earlier versions of the suggested language can also be found in (Falkman et al. 2001, Falkman and Lennartson 2001). Table II.1 briefly describes the different operators defined for the PPN language.

Table II.1: Operators in the PPN language.

Operator	Description
$P_1 \rightarrow P_2$	Process P_1 followed by process P_2
$P_1 + P_2$	Alternative choice between P_1 and P_2
$\oplus\{P_1, \ldots, P_n\}$	Arbitrary order execution of P_1, P_2, …, P_n
$P_1 \& P_2$	The start and end of P_1 and P_2 is synchronized
$\Cup\{P_1, \ldots, P_n\}$	Parallel execution of P_1, P_2, …, P_n
$P^{\uparrow}$	First event in P is executed
$P^{\downarrow}$	Last event in P is executed (arriving at the final state)
$P_1^{\uparrow} \& P_2^{\uparrow}$	The start of P_1 and P_2 is synchronized

2.1 Arbitrary order

The arbitrary order operator can be described as a set of processes that are all to be executed, but not at the same time. This means that they must be executed in a sequence, and if the order does not matter this will imply $n!$ possible alternative sequences, where n is the number of processes.

The arbitrary order between two processes, $P_1 = a$ and $P_2 = b \rightarrow c$, is demonstrated in Fig. 1.

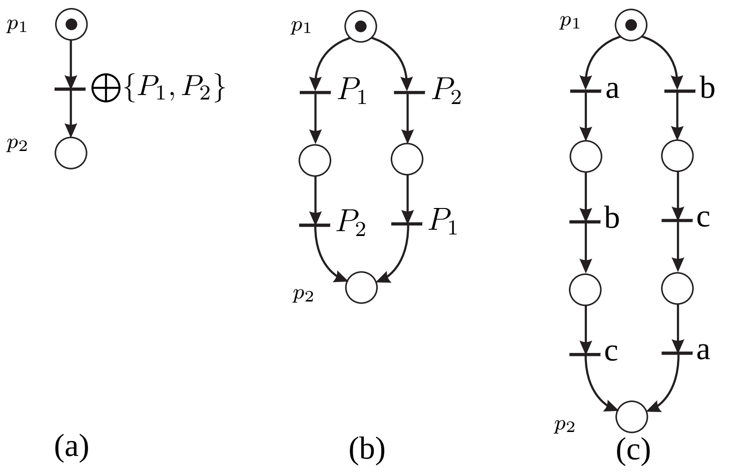


Figure 1: The process $P = \bigoplus\{P_1, P_2\}$ is in (a) and (b) given as PPN models and in (c) as a PN where $P_1 = a$ and $P_2 = b \rightarrow c$.

2.2 Synchronization

The synchronization operator & implies that one or more processes are to synchronize their first and last events respectively. Similar ideas for event synchronization can be found in (Arnold 1994). Note that the processes are not synchronized by common events. This is the opposite to Hoare's full synchronous composition (Hoare 1985) denoted $\|$, where events common to two processes P and Q must occur in both of them in order to occur in $P \parallel Q$.

The synchronized process $P_3 \& P_4$ is given as a PPN specification in Fig. 2a. The two processes $P_3 = a \rightarrow b$ and $P_4 = c \rightarrow d \rightarrow e$ are synchronized by their first and last events respectively, resulting in the specification seen in Fig. 2b. This means that two synchronized events are created, $a \& c$ and $b \& e$. Events executed in a process after the first event, and before the final event, are executed independently of the other processes. In our example in Fig. 2 this means that event d in P_4 can execute without any concern for process P_3.

2.3 Parallel

The expression $P = \sqcup\!\sqcup\{P_1, \ldots, P_n\}$ denotes a process P, which executes all processes P_i in parallel. If processes have common events these events have to be synchronized. The operator $\sqcup\!\sqcup$ is an extension of Hoares full synchronous composition (FSC) (Hoare 1985), involving the explicit process synchronization operator $\&$.

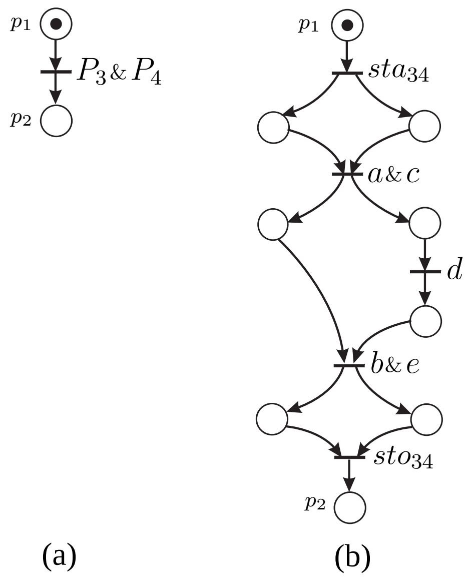


Figure 2: The process $P = P_3 \& P_4$ given in (a) as a PPN model and in (b) as a PN model (including synchronized events) using the processes $P_3 = a \rightarrow b$ and $P_4 = c \rightarrow d \rightarrow e$.

Two processes, $P_3 = d{\rightarrow}e\,\&\,f{\rightarrow}g$ and $P_4 = h{\rightarrow}e{\rightarrow}m$, have a common event e. These are to be executed in parallel (as shown in Fig. 3). The synchronized event $e\&f$ in process P_3 is required to occur at the same time as the event e in P_4.

2.4 Restrictions

The introduction of additional requirements on the execution of processes is of great importance when modelling discrete event systems in a compact, but yet readable, manner. The requirements are described by logical expressions specifying when a particular process can begin and/or end its execution.

Table II.2 summarizes the basic atomic restrictions and related state(s).

Table II.2: Restriction expressions in the PPN language.

Restriction	State(s)
$[P]$	Initial state in process P
$[P^{\downarrow}]$	Final state in process P
$[\overline{P}]$	All states except the initial state
$P_1[P_2]$	P_1 executed when P_2 is in its initial state
$P_1[P_2^{\downarrow}]$	P_1 executed when P_2 is in its final state
$P_1[P_2 \wedge \overline{P_3}]$	P_1 executed when P_2 is in its initial state while P_3 is not

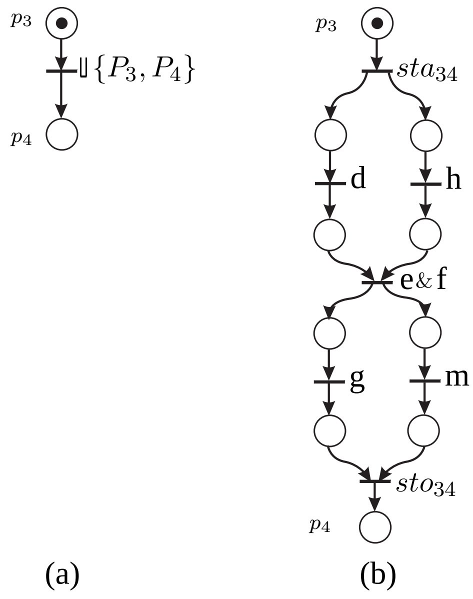


Figure 3: The process $P = \Cup\{P_3, P_4\}$ given in (a) as a PPN model and in (b) as a PN (including synchronized events).

2.5 Textual representation

A textual representation of the specifications created with the PPN language has also been developed using the extensible markup language (XML). The main reason for using XML as a textual representation of PPN models is that we want to use a well founded standard for the communication of information to other software systems.

The intended use of the PPN specification language is described in Fig. 4. This figure shows how a PPN specification, created by a user, can be represented as an XML file, which can in its turn be converted into both an automata and a STEP-AP214 representation.

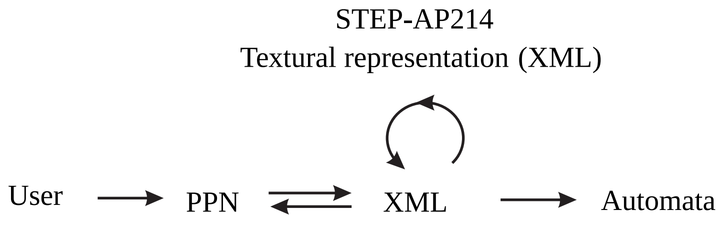


Figure 4: Relations between PPN, STEP-AP214, and finite state automata.

3 Supervisory control theory and finite state automata

In this section a brief description of both the supervisory control theory and finite state automata is provided.

3.1 Supervisory control theory

Supervisory control theory is a theory for supervisor synthesis given two types of models, the plant model and the specification model. The task of the plant model is to model all the possible events that, for example, a resource may go through, while the specification model describes the desired and allowed behavior of the plant. Supervisory control theory also defines how to treat different kinds of events, for instance controllable, uncontrollable, observable and unobservable events. By synchronizing the plant and the specification models a description of the whole closed loop system, also called total specification S_0, is generated. This total specification is then manipulated in order to remove uncontrollable states and other forbidden states, including blocking states (Ramadge and Wonham 1987). The result is the final supervisor S.

Formal high level specifications, based on the PPN language introduced in the previous section, are meant to be used as a communication link between, for instance, different departments within a company, or between supplier and costumer. It is however also possible to use these specifications for supervisor synthesis applying supervisory control theory (SCT) (Ramadge and Wonham 1987).

We have chosen to use *Supremica* (Åkesson et al. 2003) as a tool for SCT. This is a program that uses finite state automata (FSA) as its input and output language. In order to use *Supremica* for supervisor synthesis, all PPN specifications are therefore formally and automatically converted into FSA models.

3.2 Finite state automata

A finite state automaton is a structure $A = (Q, \Sigma, \delta, q_0)$ where Q is the set of finite states in the automaton. The alphabet Σ declares all events that an automaton can participate in. Some of the events in the alphabet may never be executed, an important fact when supervisor synthesis is performed. The partial transition function δ declares all defined transitions from a state via an event to another state. The initial state is given by q_0. Note that parallel behavior is not explicitly modelled using finite state automata. This is instead described indirectly using a parallel composition between two or more automata. In this paper Hoares full synchronous composition (Hoare 1985) is applied.

4 Translation of PPN to automata

In the following section the translation between a PPN model and automata models is presented. In order to describe this procedure an example is introduced. As the previously mentioned Volvo case does not involve all the operators offered by the PPN language, a fictive example including the whole range of operators will be used instead.

4.1 PPN model

Consider the PPN model in Fig. 5, where the order between a number of operation processes O_1–O_{10} is specified. The purpose of the specification is to describe when each of the processes is actually allowed to execute.

In more detail Fig. 5 describes four parallel paths followed by a final process expression.

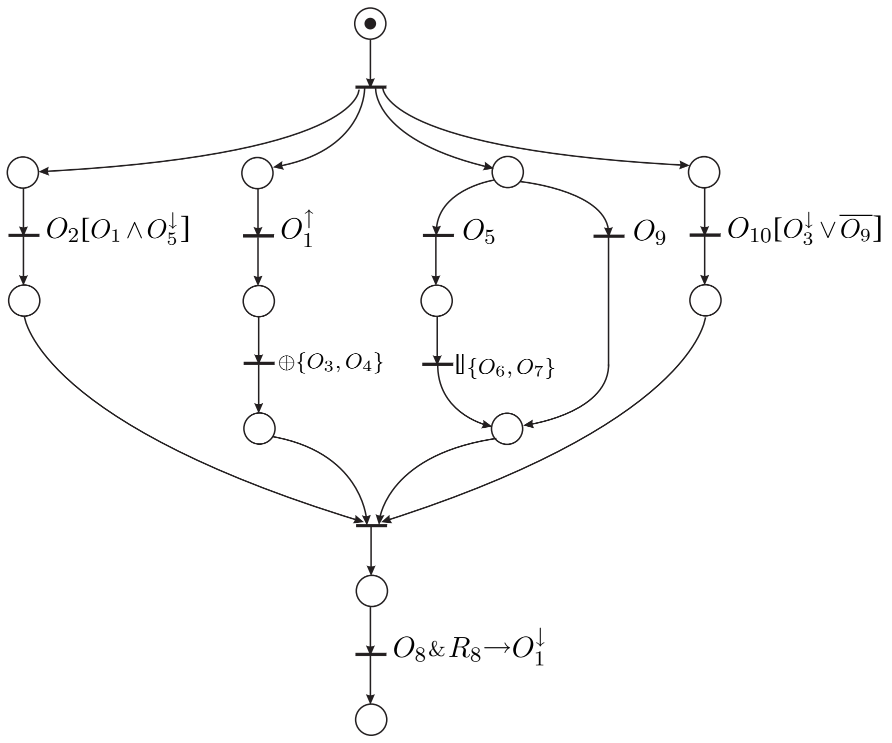

$O_2[O_1 \wedge O_5^{\downarrow}]$
$O_1^{\uparrow}$
O_5
O_9
$O_{10}[O_3^{\downarrow} \vee \overline{O_9}]$
$\oplus\{O_3, O_4\}$
$\|\{O_6, O_7\}$
$O_8 \& R_8 \rightarrow O_1^{\downarrow}$

Figure 5: Example given as a PPN specification.

Two of the parallel paths involve one operation each with restrictions on the start of their executions. Operation O_2 cannot execute before operation O_5 has finished its execution and operation O_1 has not started (initial state). Operation O_{10} is not allowed to start its execution until operation O_3 has finished its execution or operation O_9 has started its execution. The two other paths in Fig 5 involve a sequence and an alternative respectively. The sequence path specifies that operation O_1 starts its execution followed by O_3 and O_4 executed in arbitrary order. The alternative path specifies the execution of either O_9, or O_5 followed by O_6 and O_7 executed in parallel. When all four parallel paths have finished their respective transitions, the operation O_8 and process R_8 are started as well as ended at the same time. This is followed by the ending of O_1. Operation R_8 in this example represents a resource, which can be booked as well as unbooked, see Fig. 13.

4.2 Tree structure

The PPN model in Fig.5 has the tree structure in Fig. 6. This tree structure is a first step in the translation from PPN to automata.

Starting from the leaf nodes, composition operators are encountered when going up the syntax tree. The encountered operators describe how the nodes on a lower level are composed. Doing this all the way up in the syntax tree, node by node, means that the root node, symbolizing the main process, is finally reached.

There are two types of nodes in the tree structure in Fig. 6, process nodes and operator nodes. *Process nodes* involve the basic processes including optional restrictions. All other nodes involve the different operators and are also called *operator nodes*, e.g. *sequence node*, *alternative node* etc. Each node in the tree has an identity (a number), which will be used in the naming of automata models.

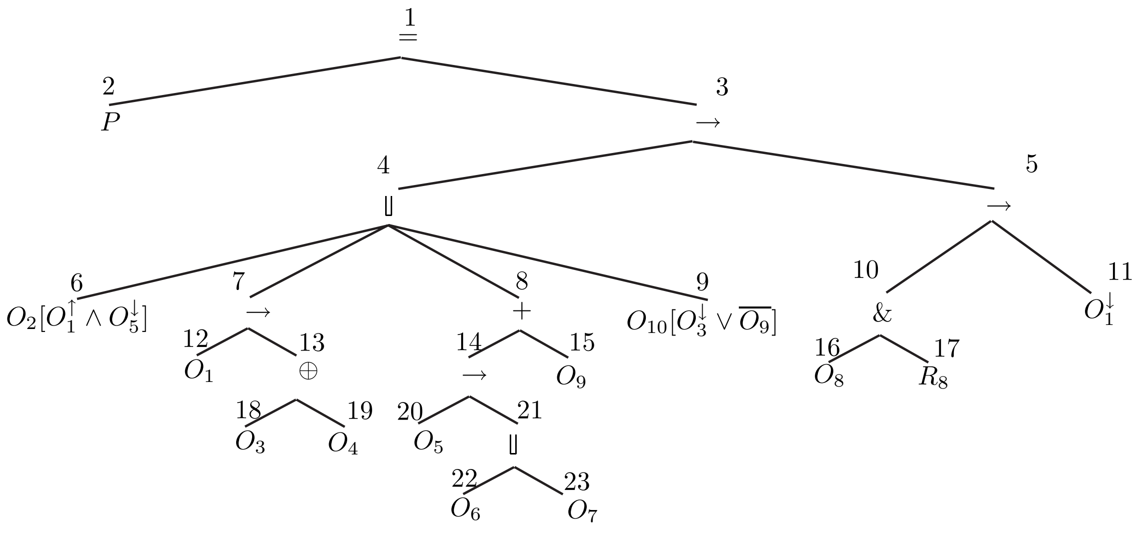


Figure 6: Tree structure for the PPN example in Fig. 5.

4.3 Informal translation from the tree structure to automata

Using the tree structure in Fig. 6 as an example, it is now informally shown how a PPN specification is converted into a finite state automata representation. In the translation from the tree structure to finite state automata we note that modular automata are preferable. The reason is that our software for synthesis and verification *Supremica* uses modular approaches to avoid the state space explosion problem for large complex systems. Two automata are created for every *process node*, while a single automaton is created for every *operator node*, except for the *parallel node* where one automaton is created for each child. Restriction expressions are not translated into ordinary automata, but modified to include actual states. The reason for this flexibility is that *Supremica* accepts extended automata, including such logical restrictions at the state transitions (similar to StateChart). Furthermore, note that all modular automata are finally connected applying Hoares full synchronous composition (FSC) (Hoare 1985).

In Fig. 7 a simple tree structure is given involving three nodes, two process nodes, m and n, and a general operator node ℓ. Node m specifies the start of process O_i using the start operator $\uparrow$. Node n specifies the execution of process O_j when process O_k is in its initial state and process O_p is in its final state. The operator node ℓ specifies any operator using the generic operator notation $*$. This tree structure will be used to describe how finite state automata are created for every type of node.

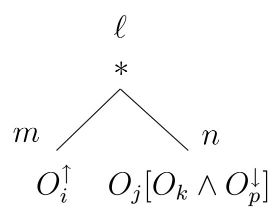


Figure 7: A tree structure with two *process nodes* and one *operator node*.

Process node

Two automata are created for every *process node*. The first automaton, called *process automaton*, see O_i and O_j in Fig. 8, includes three states and two events. The states are uniquely named using the process name together with number 1, 2, or 3, e.g. O_{j_1}. This is done because the state names in *process automata* are used when translating a restriction expression into the automata model, see automaton O_j in Fig. 8. The two events in an *process automaton* are a start event including possible restriction expression followed by a stop event for the specific process to be executed, e.g. $sta\_O_j[O_{k\_1} \wedge O_{p\_1}]$ and sto_O_j in the automaton O_j.

The second automaton created for a *process node*, called *process node automaton*, i.e. A_m and A_n in Fig. 8, includes a sequence of three or four events. A start event for the specific node, i.e. sta_m in automaton A_m, is followed by the process to be executed. This can be only one event, if the beginning or end of the process is specified using start or stop operator as for process O_i in Fig. 7. Otherwise, both the start event and the stop event for the specific process is included in the *process node automaton* followed by a stop event for the actual node.

$$\begin{tikzcd}
 & O_i & & O_j \\
{} \arrow[r] & O_{i\_1} \arrow[d, "sta\_O_i"] & {} \arrow[r] & O_{j\_1} \arrow[d, "{sta\_O_j[O_{k\_1} \wedge O_{p\_3}]}"] \\
 & O_{i\_2} \arrow[d, "sto\_O_i"] & & O_{j\_2} \arrow[d, "sto\_O_j"] \\
 & O_{i\_3} & & O_{j\_3}
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & A_m & & A_n \\
{} \arrow[r] & q0 \arrow[d, "sta\_m"] & {} \arrow[r] & q0 \arrow[d, "sta\_n"] \\
 & q1 \arrow[d, "sta\_O_i"] & & q1 \arrow[d, "{sta\_O_j[O_{k\_1} \wedge O_{p\_3}]}"] \\
 & q2 \arrow[d, "sto\_m"] & & q2 \arrow[d, "sto\_O_j"] \\
 & q3 & & q3 \arrow[d, "sto\_n"] \\
 & & & q4
\end{tikzcd}$$

(b)

Figure 8: (a) Process automata and (b) process node automata for each process node of the tree structure in Fig. 7.

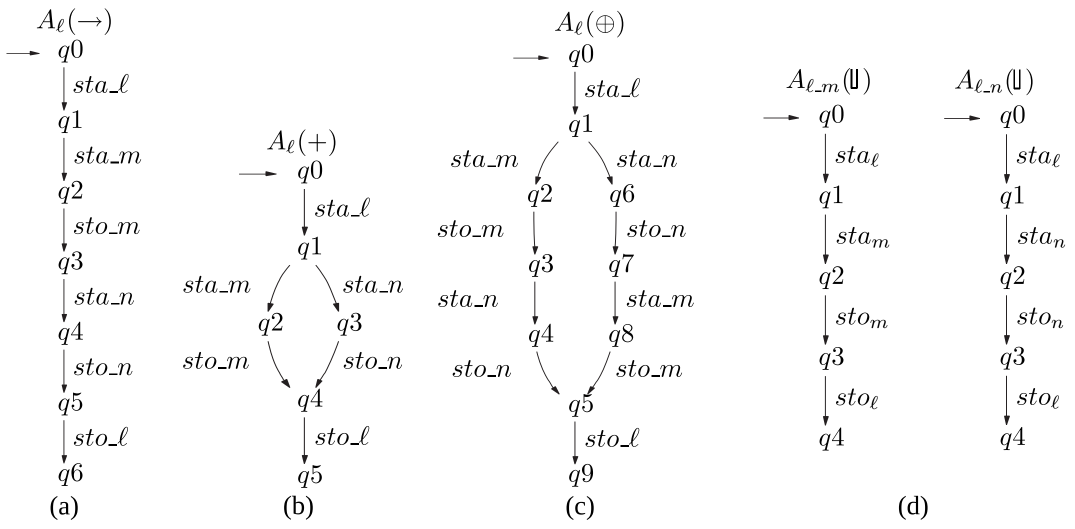


Figure 9: Automata models for the operator node in Fig. 7 where node ℓ is (a) a *sequence node*, (b) an *alternative node*, (c) an *arbitrary order node*, and (d) a *parallel node*.

Operator node

The automaton for the *operator nodes* consist of three parts, a start event followed by an automata model for the actual operator and then a stop event.

Sequence node A *sequence node* automaton is given in Fig. 9(a) for node ℓ in Fig.7 with $*$ equal to $\rightarrow$. An automaton for a sequence node always includes six events, beginning with the start event of the sequence node. This event is followed by the start and stop events of its first child, sta_m and sto_n in Fig. 9(a), followed by the start and stop events of the second child node, sta_n and sto_n. The sequence is ended by executing its stop event.

Alternative node A corresponding *alternative node* automaton is given in Fig. 9(b). This automaton, A_ℓ, begins with a start event sta_ℓ and ends with a stop event sto_ℓ. The start event of the alternative node is followed by the start and stop events of either child m or child n.

Arbitrary order node The *arbitrary order node* can have more than two child nodes, which means that the corresponding node automaton does not have a fixed number of states. After the start event follows all possible permutations of the child nodes start and stop events. Fig. 9(c) shows an automaton for the arbitrary order node in Fig. 7 with $*$ equal to $\oplus$.

Parallel node A set of automata is created for a *parallel node*, one for each of its child nodes. All these automata include a sequence of four events with the same first and last events, which are the start and stop events for the *parallel node*. Two automata for the *parallel node* in Fig. 7, with $*$ equal to $\Cup$, is given in Fig. 9(d). The common start event in each of these automata models is followed by the start and stop events of its child node. The parallel behavior is finally obtained by connecting all involved automata by Hoares full synchronous composition (FSC). This means explicitly that the common start and stop events in the parallel node automata are synchronized.

Synchronization node The synchronization operator $\&$ is mainly used between individual basic processes, e.g. between a number of resources and an operation $(R_i \& R_j \& O_k)$. To avoid complications in the automata translation we therefore restrict the children nodes of the synchronization operator to involve only process nodes. Multiple process synchronization means that there is one child node for each involved process. Fig. 10 shows an operator automaton for the synchronization operator in Fig. 7 with $*$ equal to $\&$. The start event sta_ℓ is followed by a synchronized event between the start events of the actual processes to be executed, i.e. $sta\_O_i \& sta\_O_j[O_{k\_1} \wedge O_{p\_1}]$. This is then followed by the synchronized event of the stop events, in this case only the event sto_{O_j} since the first synchronized process $O_i^{\uparrow}$ only includes one event.

It is not possible to have any synchronized events in the automata when supervisor synthesis is performed using Supremica. Since the synchronized events are preserved when the PPN models are converted into FSA, it is also necessary to finally perform a relabelling of the created automata to avoid such synchronized events, e.g. $a \& b$. This relabelling is described in further detail in (Falkman and Lennartson 2005a).

$$\begin{tikzcd}
 & A_\ell \\
{} \arrow[r] & q0 \arrow[d, "sta_\ell"] \\
 & q1 \arrow[d, "sta\_O_i \& sta\_O_j[O_{k\_1} \wedge O_{p\_3}]"] \\
 & q2 \arrow[d, "sto\_O_j"] \\
 & q3 \arrow[d, "sto_\ell"] \\
 & q4
\end{tikzcd}$$

Figure 10: Automaton for the operator node in Fig. 7 where node ℓ is a *synchronization node*.

Example: In Fig. 11 a small part of the tree structure in Fig. 6 is given, representing the expression $(O_5 \rightarrow\Vert \{O_6, O_7\}) + O_9$. The corresponding eight automata models are also shown in Fig. 11. Note that the *process automaton* for each process node have been left out.

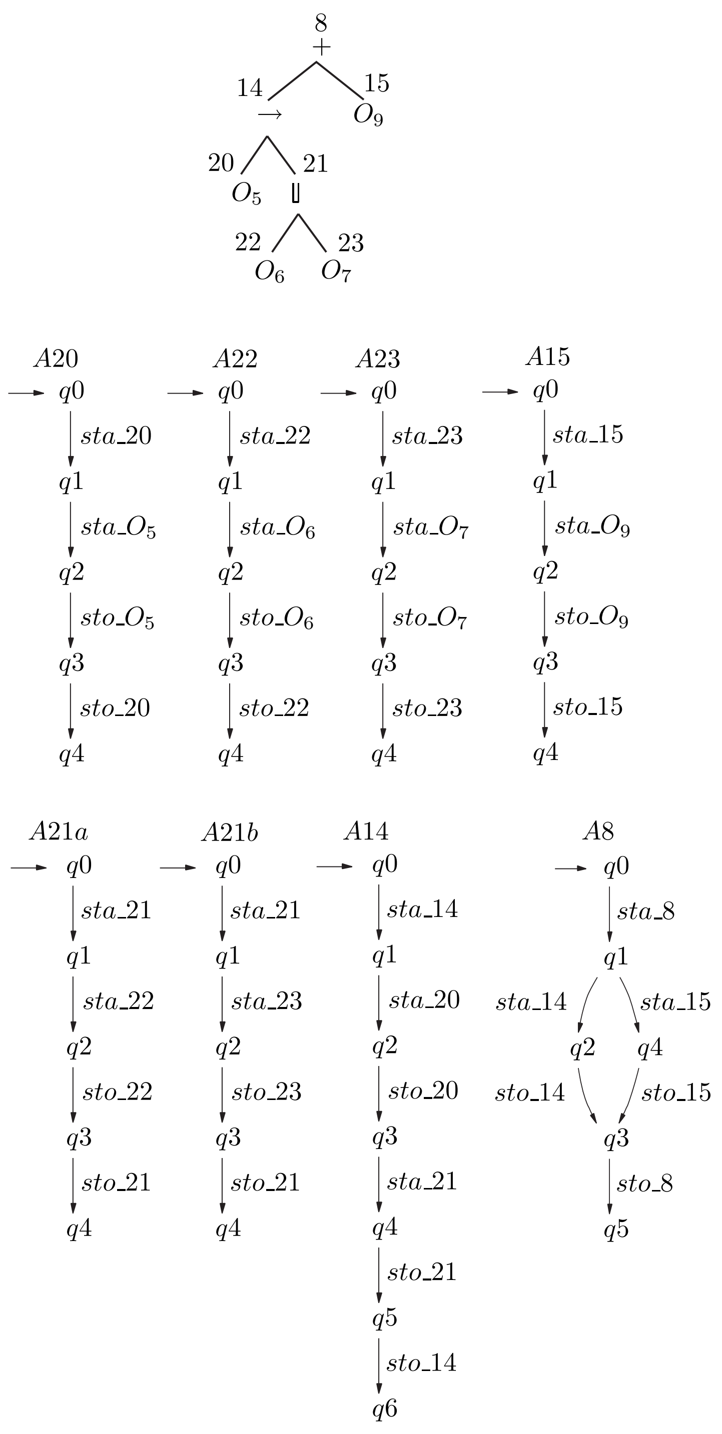


Figure 11: A small part of the tree in Fig. 6, representing the expression $(O_5 \rightarrow \Cup\{O_6, O_7\}) + O_9$ together with corresponding automata models. One automaton is created for each node. Each individual automaton begins with a start event and ends with a stop event for that specific node.

4.4 Formal translation

A translation between two different languages can be defined more formally by the use of an algorithm. In this section part of the algorithm necessary for the translation from the

PPN language to a finite state automata representation is presented. The algorithm includes a recursive function, which goes through the entire tree structure. It creates an automaton for each node, starting at the root node and proceeding down in the tree, node by node.

The function *visit* given in Algorithm. 1 describes how the conversion from a PPN representation to an automata representation is performed.

A *process node* has three attributes, node identity *id*, process identity *pid*, and operator *op*. The attribute *pid* defines the actual process to be executed and the attribute *op* describes if the start or stop operator is used. An *operator node* also has three attributes, node identity *id*, *type*, and children *ch*. What kind of operator is given by the attribute type and the attribute children refers to the current nodes children.

Three functions, *CreateSynchronizationAutomaton*, *CreateAlternativeAutomaton*, and *CreateArbitraryAutomaton* are used in Algorithm 1. Function *CreateSynchronizationAutomaton* creates a set of automata which defines the synchronization as in Fig. 10, and *CreateAlternativeAutomaton* creates an automaton describing the alternative execution of two children and returns an automaton as in Fig. 9(b). Function CreateArbitraryAutomaton is a function which returns an automaton describing the arbitrary order execution of involved children as in Fig.9(c).

An automaton is created for the *synchronization node*, in Algorithm 1. A sequence of four events with the same first and last events, which are the start and stop events for the *synchronization node* is created. The start event is followed by a synchronized event between the start events of the actual processes to be executed. This is then followed by the synchronized event of the stop events. The case when the start or stop operator is used, e.g. $O_i^{\uparrow}$, only the first transition will include a synchronized event since $O_i^{\uparrow}$ only includes one event, cf. Fig. 10. If all processes are specified using the start or stop operator only one transition, with a synchronized event, is created.

A set of automata is created for a *parallel node*, one for each of its child nodes. All these automata include a sequence of four events with the same first and last events, which are the start and stop events for the *parallel node*. The common start event in each of the automata models is followed by the start and stop events of its child node.

5 Industry example

The proposed language may be used for modelling of flexible manufacturing systems (FMS), especially resource allocation systems. In a FMS a range of different products make use of a number of different resources in order to perform a specified sequence of operations. These operations have to be executed in a certain order using specific resources. All products may want to use the same resources, and it is therefore necessary to control that only one product will use a specific resource at a time. This kind of control system is called a resource allocation system.

The following example describes how a resource allocation system may easily be modelled using the suggested PPN language. The plant to be controlled is a robot cell at Volvo Car Corporation, Torslanda, Sweden, see Fig. 12. A detailed description of this cell is presented in (Richardsson 2005).

The robot cell in Figure 12 consists of four robots R_1-R_4, two Fixtures F_1-F_2, two turntables T_1-T_2, and a conveyor C_1. Due to symmetry only the resources below the conveyor, in Figure 12, is considered in the example.

Algorithm 1: `visit`

```
    Input       : Node n
    // Create automata for current node;
1   case n.type is process
        // Create a process automaton, cf. Fig. 8(a);
2       createState(n.pid + "_1", initial);
3       createState(n.pid + "_2");
4       createState(n.pid + "_3");
5       createTransition(n.pid + "_1", n.pid + "_2", "sta_" + n.pid);
6       createTransition(n.pid + "_2", n.pid + "_3", "sto_" + n.pid);
        // Create a process node automaton, cf. Fig. 8(b);
        if n.op = null then
7           createState("q0", initial);
            for i = 1 ... 4 do
8               createState("q" + str(i));
9           createTransition("q0", "q1", "sta_" + n.id);
10          createTransition("q1", "q2", "sta_" + n.pid);
11          createTransition("q2", "q3", "sto_" + n.pid);
12          createTransition("q3", "q4", "sto_" + n.id);
        if n.op ≠ null then
13          createState("q0", initial);
            for i = 1 ... 3 do
14              createState("q" + str(i));
15          createTransition("q0", "q1", "sta_" + n.id);
            if n.op = ↑ then
16              createTransition("q1", "q2", "sta_" + n.pid);
            if n.op = ↓ then
17              createTransition("q1", "q2", "sto_" + n.pid);
18          createTransition("q2", "q3", "sto_" + n.id);
19  case n.type is sequence
        // Create an automaton specifying sequence, cf. Fig.9(a);
20      createState("q0", initial);
        for i = 1 ... 6 do
21          createState("q" + str(i));
22      createTransition("q0", "q1", "sta_" + n.id);
23      createTransition("q1", "q2", "sta_" + n.ch[1].id);
24      createTransition("q2", "q3", "sto_" + n.ch[1].id);
25      createTransition("q3", "q4", "sta_" + n.ch[2].id);
26      createTransition("q4", "q5", "sto_" + n.ch[2].id);
27      createTransition("q5", "q6", "sta_" + n.id);
28  case n.type is alternative
29      CreateAlternativeAutomaton;
        // Create an automaton specifying alternative, cf. Fig.9(b);
30  case type is arbitrary order
31      CreateArbitraryAutomaton;
        // Create an automaton specifying arbitrary order, cf. Fig.9(c);
32  case n.type is synchronization
33      CreateSynchronizationAutomaton;
        // Create an automaton specifying synchronization, cf. Fig.10;
34  case n.type is parallel
35      for i = 1 ... number of children to current node do
36          createState("q0", initial);
            for j = 1 ... 4 do
37              createState("q" + str(j));
38          createTransition("q0", "q1", "sta_" + n.id);
39          createTransition("q1", "q2", "sta_" + n.ch[i].id);
40          createTransition("q2", "q3", "sto_" + n.ch[i].id);
41          createTransition("q3", "q4", "sto_" + n.id);
42  //Add current automaton to a set of created automata;
43  for i = 1 ... number of children to current node do
44      visit(n.ch[i]);
```

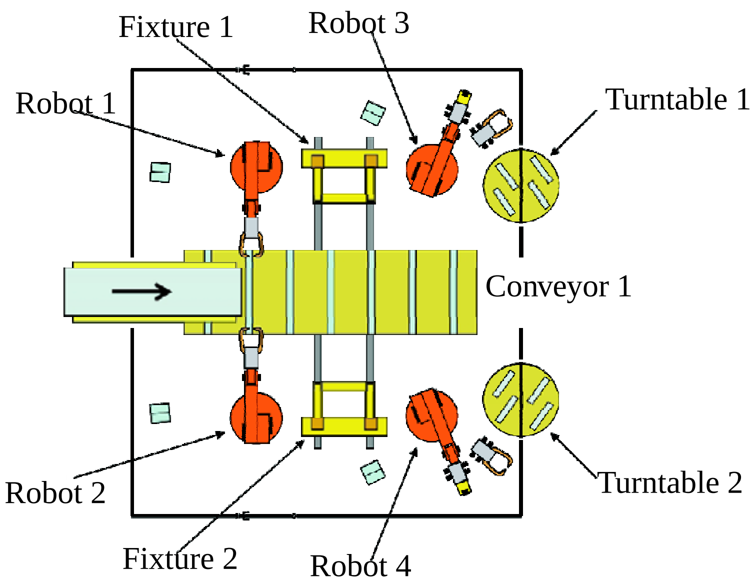


Figure 12: The example cell with four robots, two fixtures, and two turntables, and a conveyor.

5.1 Plant model

Each resource in the robot cell is modelled as a recursive process, see Fig. 13(a), and R_ℓ represents any of the resources. Assume that $R_\ell^i = b_\ell^i \to u_\ell^i$, where b_ℓ^i represents the booking of resource R_ℓ by product i and event u_ℓ^i represents the corresponding unbooking event. A corresponding PN for R_ℓ where $R_\ell^1 = a_\ell^1 \to b_\ell^1$ and $R_\ell^n = a_\ell^n \to b_\ell^n$ is given in Fig. 13(b).

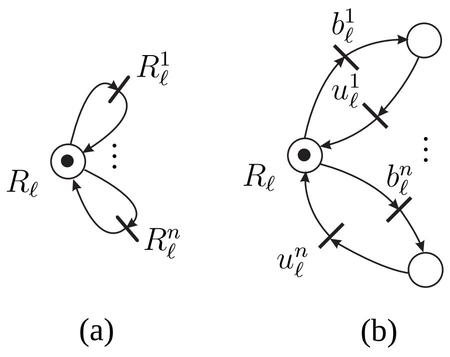


Figure 13: Resource model R_ℓ given as (a) a PPN model and (b) a PN. The resource can be booked by a number of routing specifications $1, \ldots, n$.

5.2 Specifications

The task of the example cell in Fig. 12 is to weld a plate to the side of the floor of the car, underneath the doors. This operation is to be executed on both car models produced in the cell, i.e. Volvo V70, and S80. During a work cycle Robot 4 picks a part from the rack on Turntable 2 and then places it in Fixture 2. Simultaneously Robot 2 starts to weld previously loaded, but not completely welded, parts of the body. Robot 4 changes tool from gripper to

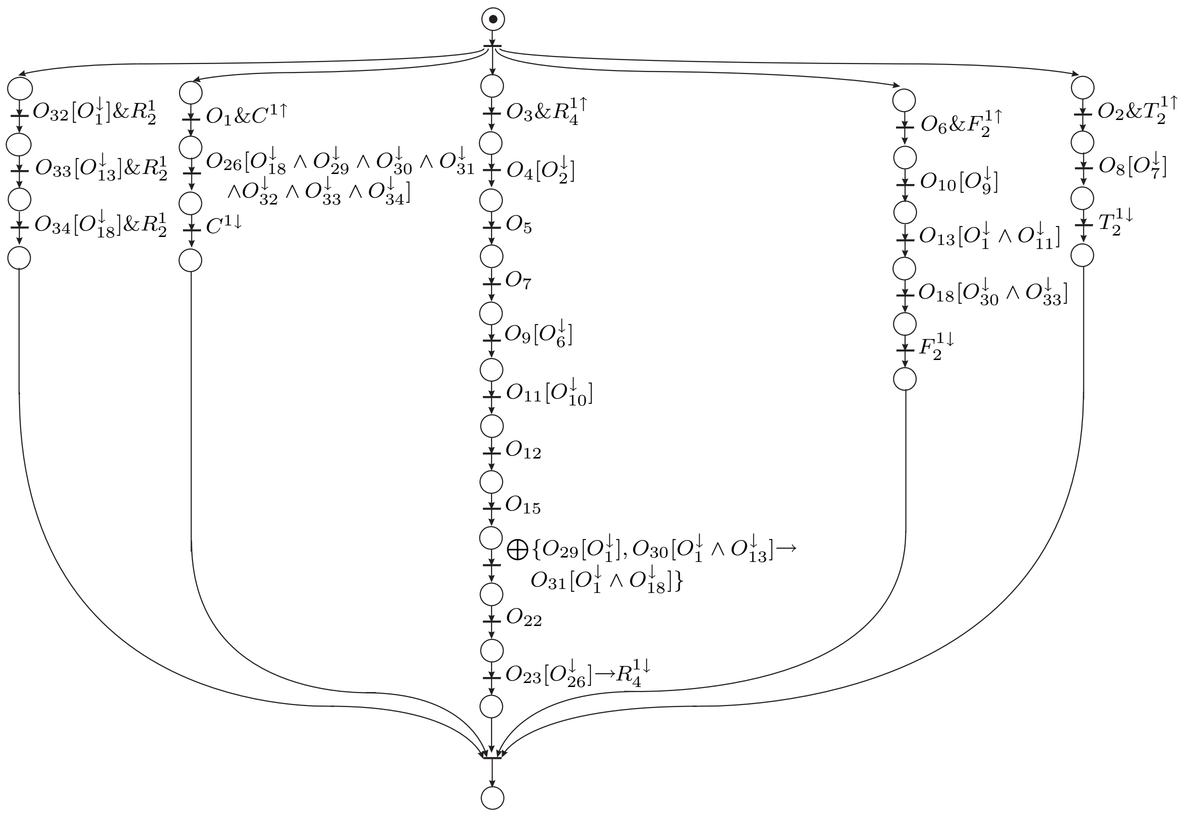

$O_{32}[O_1^{\downarrow}]\&R_2^1$
$O_{33}[O_{13}^{\downarrow}]\&R_2^1$
$O_{34}[O_{18}^{\downarrow}]\&R_2^1$
$O_1\&C^{1\uparrow}$
$O_{26}[O_{18}^{\downarrow}\wedge O_{29}^{\downarrow}\wedge O_{30}^{\downarrow}\wedge O_{31}^{\downarrow}$
$\wedge O_{32}^{\downarrow}\wedge O_{33}^{\downarrow}\wedge O_{34}^{\downarrow}]$
$C^{1\downarrow}$
$O_3\&R_4^{1\uparrow}$
$O_4[O_2^{\downarrow}]$
O_5
O_7
$O_9[O_6^{\downarrow}]$
$O_{11}[O_{10}^{\downarrow}]$
O_{12}
O_{15}
$\bigoplus\{O_{29}[O_1^{\downarrow}], O_{30}[O_1^{\downarrow}\wedge O_{13}^{\downarrow}]\rightarrow$
$O_{31}[O_1^{\downarrow}\wedge O_{18}^{\downarrow}]\}$
O_{22}
$O_{23}[O_{26}^{\downarrow}]\rightarrow R_4^{1\downarrow}$
$O_6\&F_2^{1\uparrow}$
$O_{10}[O_9^{\downarrow}]$
$O_{13}[O_1^{\downarrow}\wedge O_{11}^{\downarrow}]$
$O_{18}[O_{30}^{\downarrow}\wedge O_{33}^{\downarrow}]$
$F_2^{1\downarrow}$
$O_2\&T_2^{1\uparrow}$
$O_8[O_7^{\downarrow}]$
$T_2^{1\downarrow}$

Figure 14: A relation of operation ROP specification, given as a PPN model, for production of a Volvo V70 in a robot cell at Volvo Car Corporation, Torslanda, Sweden.

weld gun, the fixture positions the plate on the body, and both robot 2 and 4 weld the new part to the body.

A relation of operation (ROP) (Richardsson 2005) specification is presented in Table II.3. A ROP specification is an progress specification specifies what all the operations do and which resources they require. It is also specified if an operation have predecessors, which are operations that have to have finished their execution before the actual operation starts. The ROP in Table II.3 is given as a PPN model in Fig. 14. This PPN model uses the PN constructs in order to create five parallel sequences, one for each resource. A sequence of operations is also identified, for each resource, by looking at the predecessors in Table II.3.

A single product, V70, is using the resources in this example, which means that a single process R_ℓ^1, cf. Fig. 13, is created for each resource. Every operation requires a single resource, which is specified using the synchronization operator $\&$ in Fig. 14. The Conveyor, Turntable 2, Fixture 2, and Robot 4, are booked by the first operation that requires them and are unbooked after the last operation that requires them. Robot 2 is both booked and unbooked by every operation using it. This is advantageous if more than one routing specification at a time wants to use the specific resource.

The PPN model in Fig. 14 uses the arbitrary order operator as well as restriction expression in order to achieve a compact model without sacrificing readability. The restriction expressions reduce the number of PN arcs crossing the parallel paths and thereby reducing the complexity and increasing the readability of the PN. In the next subsection it is shown how the achieved ROP specification can be used for supervisor synthesis.

operation	resource	comment	predecessor	product
1	conveoyr	move in floor	-	all
26	conveoyr	move out floor	18,29,30,31, 32,33,34	V70
6	fixture 2	ready to receive part?	-	all
10	fixture 2	clamp part	9	all
13	fixture 2	go to work position	1,11	all
18	fixture 2	go to home position	30,33	V70
32	robot 2	weld job 1	1	V70
33	robot 2	weld job 2, weld part hold by fixture	1,13, 32	V70
34	robot 2	weld job 3, weld new part without fixture	1,18, 33	V70
3	robot 4	get gripper 1	-	all
4	robot 4	go to rack	2,3	all
5	robot 4	pick part	4	all
7	robot 4	go to fixture	5	all
9	robot 4	put part in fixture	6, 7	all
11	robot 4	go out of fixture	10	all
12	robot 4	go to tool change pos	11	all
15	robot 4	get weld gun	12	all
22	robot 4	go to tool change pos	29,30,31	all
23	robot 4	get gripper 2	26	all
29	robot 4	weld job 4	1,15	V70
30	robot 4	weld job 5, weld part hold by fixture	1,13,15	V70
31	robot 4	weld job 6, weld new part without fixture	1,15,18	V70
2	turntable 2	part ready?	-	all
8	turntable 2	rotate turntable	7	all

Table II.3: A description of operations, required resources, and predecessors for Volvo v70.

5.3 Supervisor synthesis

A supervisor synthesis include the plant models and the specifications, see Fig. 15. The resources, cf. Fig. 13, specify what is possible to do in the system and thus constitute the plant models. The specification models describe both what we *want to do* (progress specification) and what we *must* or *must not do* (high level safety specification). The high level safety specifications restrict the possible operation sequences in the ROP. These safety specifications are the result of a *specification synthesis* (Andersson et al. 2005), see Fig. 15, performed on detailed operation descriptions called execution of operations (EOP) and interlocks (IL). An EOP specifies a sequence of events, which correspond to state changes in one or more of the used resources. Interlocks are safety requirements that prevent damage of the resources and these are specified as boolean expressions of resource and/or operation states. One or more IL expressions are associated with an event in an EOP. More detailed description of EOP and IL can be found in (Richardsson 2005).

Both the EOP and the IL are described on a lower level than the ROP and it is therefore necessary to produce high level specifications (Andersson et al. 2005), which is done by performing the *specification synthesis*. In this synthesis, ILs are processed together with the EOPs, and the ILs are transformed into restrictions on the operation sequences in the ROP. A supervisor synthesis can now be performed between the ROP, high level safety specifications, and the plant, resulting in a sequence of operations SOP. This SOP can be used as a supervisor for the system.

In order to perform supervisor synthesis, using *Supremica*, both the resource models in Fig. 13 as well as the PPN specification in Fig. 14 are converted into finite state automata. The conversion is followed by a relabelling, since synchronized events are preserved

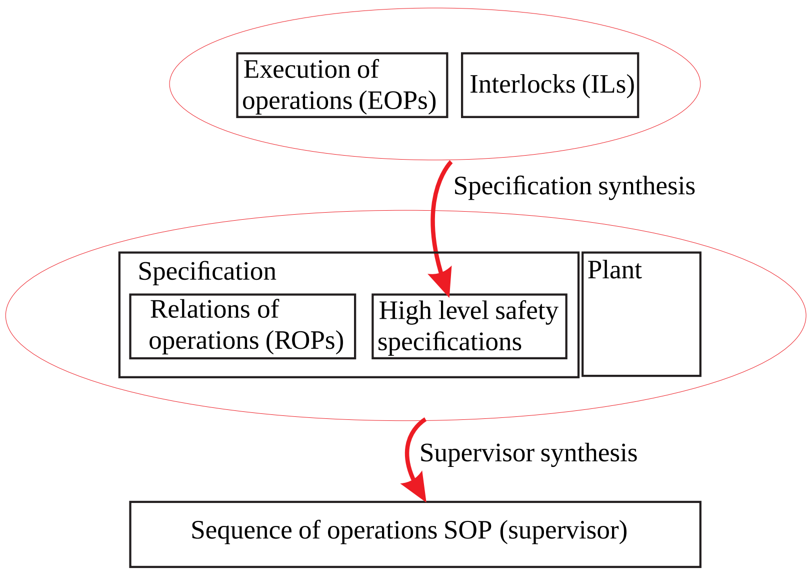

Execution of operations (EOPs)
Interlocks (ILs)
Specification synthesis
Specification
Relations of operations (ROPs)
High level safety specifications
Plant
Supervisor synthesis
Sequence of operations SOP (supervisor)

Figure 15: Different models included in the specification and supervisor synthesis.

when the PPN models are converted into FSA. This relabelling is described in further detail in (Falkman and Lennartson 2005a).

The supervisor synthesis use a modular approach (Åkesson, Flordal and Fabian 2002b) in order to handle the state explosion that arise when working with large complex systems. This modular approach can, however, be insufficient and then the supervisor synthesis can be performed using BDDs (Vahidi, Lennartson and Fabian 2005).

6 Conclusion

The PPN language defines an algebra that is combined with ordinary labelled PNs in order to realize a powerful specification language for discrete event systems, and especially flexible manufacturing systems. The present paper has shown how the PPN language can be successfully used for the specification of such systems. This was achieved by providing a real industry case involving the specification of a product to be manipulated within a robot cell. The included example has shown that the PPN language realizes concise descriptions of large complex systems, which is crucial for readability and understanding.

A translation method has also been presented, which automatically translates the PPN models into finite state automata in order to use an existing tool, *Supremica*, for supervisor synthesis.

References

Åkesson, K., Fabian, M., Flordal, H. and Vahidi, A. (2003). Supremica - a tool for verification and synthesis of discrete event supervisors, *11th Mediterranean Conference on Control and Automation*, Rhodos, Greece.

Åkesson, K., Flordal, H. and Fabian, M. (2002). Exploiting modularity for synthesis and verification of supervisors, *Proc. of the IFAC World Congress on Automatic Control.*, Barcelona, Spain.

Andersson, K., Richarsson, J., Lennartsson, B. and Fabian, M. (2005). Hierarchical control applying information reuse and supervisor synthesis, *To be submitted to Transactions on Automation Science and Engineering* .

Arnold, A. (1994). *Finite Transition Systems: Semantics of Communicating Systems*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Best, E., Devillers, R. and Koutny, M. (1998). Petri nets, process algebras and concurent programming languages, *Proc of ICM'98*, Berlin, Germany.

Best, E., Devillers, R. and Koutny, M. (2001). *Petri net algebra*, EATCS monographs on theoretical computer science, Springer, Berlin.

Best, E., Devillers, R. and Koutny, M. (2002). The box algebra = petri nets + process expressions, *Information and Computation* (178): 44–100.

Bloom, B., Cheng, A. and Dsouza, A. (1997). Using a protean language to enchance expressiveness in specification, *IEEE Transactions on Software Engineering* **23**(4): 224–234.

Brinksma, E. (1995). Performance and formal design: a process algebraic perspective, *Proc. of Sixth International Workshop on Petri Nets and Performance Models*, IEEE, Durham, NC USA, pp. 124 – 125.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, Kluwer Academic Publishers.

Degano, P., DeNicola, R. and Montanari, U. (1987). Ccs is an (augmented) contact-free c/e system, *in* E. M. Venturini Zilli (ed.), *Mathematical Models for the semantics of Parallelism*, Vol. Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 144–165.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P. and Lennartson, B. (2005). A high level specification language based on process algebra and petri nets, *To be submitted to Transactions on Automation Science and Engineering* .

Falkman, P., Lennartson, B. and Tittus, M. (2001). Modeling and specification of discrete event systems using combined process algebra, *Proc. of 2001 IEEE/ASME Advanced Intelligent Mecatronics*, COMO, Italy.

Falkman, P., Nielsen, J. and Lennartson, B. (2003). Automatic generation of object models for process planning and control purposes using an international standard for information exchange, *Proc. of SCI 2003*, Orlando, Florida, USA.

Harel, D., Pnueli, A., Schmidt, J. and Sherman, R. (1987). On the formal semantics of statecharts., *Proc. of Symposium on Logic in Computer Science.*, pp. 55–64.

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Series in Computer Science, Addison-Wesley.

ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles (1994). ISO standard.

Jmaiel, M. (2000). A unified algebraic framework for specifying communication protocols, *Proc. of Third international Conference on Formal Engineering Methods*, York UK, pp. 57–65.

Kozen, D. (1997). *Automata and Computability*, ISBN 0-387-94907-0, Springer-verlag New York, inc.

Mayr, R. (1997). Combining petri nets and pa-processes, *Theoretical Aspects of Computer Software (TACS'97), volume 1281 of Lecture Notes in Computer Science*, Sendai, Japan.

Milner, R. (1980). *A Calculus of Communicating Systems*, Vol. Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg New York.

Olderog, E.-R. (1991). *Nets, Terms and Formulas*, Cambridge University Press, Trumpington Street, Cambridge CB2 1RP, Great Britain.

Pena, M. and Cortadella, J. (1996). Combining process algebras and petri nets for the specification and systethis of asynchronious circuits, *Proc of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Fucushima, Japan.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes, *SIAM J. Control Optim.* **25**(1): 206–230.

Rescher, N. and Urquhart, A. (1971). Temporal logic, *Springer-Verlag, New York* .

Richardsson, J. (2005). *Development and Verification of Control Systems for Flexible Automation*, Licentiate thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 015.

Vahidi, A., Lennartson, B. and Fabian, M. (2005). Efficient supervisory synthesis of large systems, *Control Engineering Practice*. Accepted.

specification of a batch plant

Submitted to: *Journal of Control Engineering Practice*:

Specification of a Batch Plant using Process Algebra and Petri Nets

P. Falkman[†] and B. Lennartson[†] and M. Tittus[*]
[†]Signals and Systems
Chalmers University of Technology

[*]School of Engineering, University of Borås

The focus of the present paper is on the specification of routing and resource allocation systems. Such systems can be described as a set of shared resources and a set of products. The products utilize the resources in order to be manipulated according to a certain specification. This product specification consists of a set of operations that are to be executed in a certain order by specific resources. This results in a desired product route through the resource system, and hence the product specification is also called a *routing specification*. The process algebra Petri net (PPN) formalism, i.e. a combination of Petri nets and process algebra, implies efficient and less complex models for routing specifications compared to PNs and automata descriptions. The aim of this paper is to show how the PPN language, can be used in order to simplify the specification of desired routes of the chemical batch process.

1 Introduction

Due to the high costs of modifying and changing system implementations, the ability to model and simulate systems before they are implemented is becoming more and more important. The focus in this paper is on concurrent systems that may be modelled as discrete event systems (DES) (Cassandras and Lafortune 1999), especially routing and resource allocation systems (Åkesson 2002). Such systems may be described as a set of shared resources and a set of products. The products utilize the resources in order to be manipulated according to a certain specification. This product specification consists of a set of operations that are to be executed in a certain order by specific resources. The first operation has to be performed in one resource and the second one in another resource etc. This results in a desired product route through the resource system, and hence the product specification is also called a *routing specification*.

In order to synchronize the product utilization of the shared resources available, a *supervisor* is required. This supervisor is automatically constructed and adapted to the current resource/routing information. From a user point of view the basic idea is that the products are to route themselves through the resource system. In this perspective the supervisor only prevents the products from visiting undesirable states.

The present paper uses a high level language called process algebra Petri nets (PPN), presented in (Falkman and Lennartson 2005a), in order to simplify the specification of desired routes. This is not a first attempt at combining Petri nets and process algebra. Previous examples are, for instance; PAN (process algebra and Petri nets) (Mayr 1997) and Petri nets, process algebras and concurrent programming languages, (Best et al. 1998, Best et al. 2002). These languages focus on the introduction of Petri net concepts in process algebra. Methods have also been developed in order to represent process algebra programs by Petri nets e.g., (Rondogiannis and Cheng 1994, Olderog 1987). A combination of process algebra and Petri nets that focuses on parallel composition has also been defined for the specification and synthesis of asynchronous circuits, (Pena and Cortadella 1996). Another language that combines Petri nets and process algebra is given in (Basten 1998) where the focus is on a method supporting compositional design. The present paper shows how Petri nets and process algebra can be combined in order to achieve a modelling language that uses the graphical advantages from Petri nets and the powerful modelling features from process algebra. Hence, the focus is on the introduction of process algebra constructs in Petri nets instead of the opposite.

Previous work has defined the PPN language in detail (Falkman and Lennartson 2005a) and its relation to the international standard for information exchange STEP has been shown in (Falkman et al. 2004). It has also been shown how PPN specifications can be formally converted into finite state automata in (Falkman and Lennartson 2005b). The aim of the present work is to show how the PPN language can be utilized in order to simplify the specification of desired routes of a chemical batch process. The following batch plant example will be used and developed throughout the paper to illustrate the presented ideas.

Example 12 – The batch plant

We start with the batch plant shown in Fig. 1. It consists of twelve processors, three supply tanks (T_1 to T_3) containing raw materials M_1, M_2 and M_3, respectively, three reactors (P_{19} to P_{21}), two separation filters (P_{22} and P_{23}), and the four storage tanks (P_{24} to P_{28}). It is assumed that all outlets of the supply tanks can be used simultaneously. A pipeline system equipped with eighteen valves V_1 to V_{18} connects these processors.

□

The process algebra Petri net (PPN) language is described in Section 2, after that general models for resources and routing specifications are given in Section 3. Split and join operations are introduced in Section 4 and a larger example is given in Section 5.

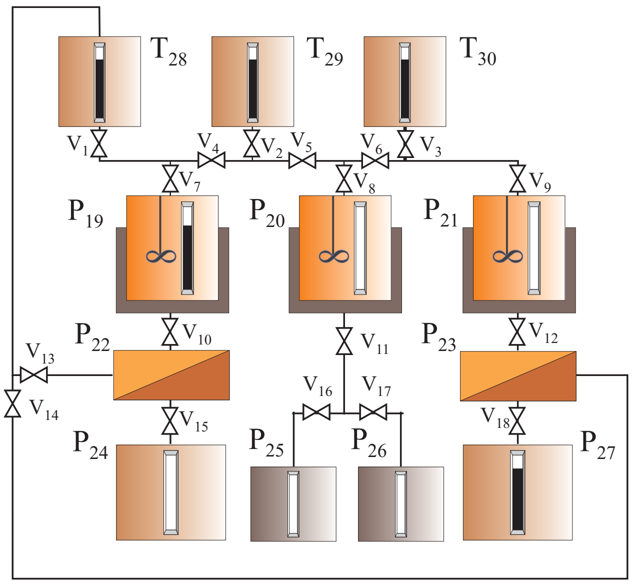


Figure 1: A batch Plant.

2 Process algebra Petri net (PPN)

In order to achieve a high level formalism for creating the routing specification, a language which combines ordinary labelled safe Petri nets (PNs) and process algebra will be used. The suggested formalism called process algebra Petri net (PPN), introduced in (Falkman and Lennartson 2005a), uses the graphical representations of Petri nets and the compact representations of process algebra in order to realize a language that may deliver concise descriptions of complex systems. In this section a brief description of some of the operators defined for the PPN language is given. A method is also presented which allows us to restrict when a process may execute. This is very useful when modelling split and join processes in Section 4.

A more formal definition of the PPN language is given in (Falkman and Lennartson 2005a). Operator laws together with the relation between the PPN models and PNs are also given. A mapping method and an algorithm to automatically convert PPN models into finite state automata (FSA) (Hopcroft et al. 2001) are given in (Falkman and Lennartson 2005b). Earlier versions of the suggested language can also be found in (Falkman et al. 2001, Falkman and Lennartson 2001). A selection of the operators, defined in the PPN language, are used in this paper and these are presented in Table III.1.

2.1 Alternative

The alternative between three processes $P_1 = a$, $P_2 = b \rightarrow c$, and $P_3 = d \rightarrow e$ is modelled in Fig. 2 both as PPNs and PN.

2.2 Synchronization

The synchronization operator & implies that one or more processes are to synchronize their first and last events respectively. Similar ideas for event synchronization can be found in

Table III.1: Operators and restriction expressions in the PPN language.

Operator	Description
$P_1 \to P_2$	Process P_1 followed by process P_2
$P_1 + P_2$	Alternative choice between P_1 and P_2
$P_1 \& P_2$	The start and end of P_1 and P_2 is synchronized
$\Vert\{P_1, \ldots, P_n\}$	Parallel execution of P_1, P_2, …, P_n
Restriction	**State(s)**
$[P]$	Initial state in process P
$[\overline{P}]$	All states except the initial state
$P_1[P_2]$	P_1 executed when P_2 is in its initial state
$P_1[P_2 \wedge \overline{P_3}]$	P_1 executed when P_2 is in its initial state while P_3 is not

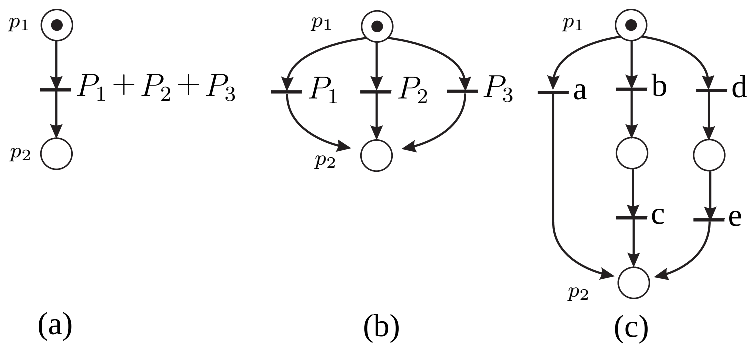


Figure 2: The process $P = P_1 + P_2 + P_3$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a$, $P_2 = b \to c$, and $P_2 = d \to e$.

(Arnold 1994). The processes are not synchronized by common events, which is the opposite to Hoare's full synchronous composition (Hoare 1985). If a synchronized process only involves an atomic process, then this will be synchronized with the first event of the other synchronized processes, and then taking no part in the synchronization of the last events. The explicit synchronization of two processes $P = P_1 \& P_2$ where $P_1 = a$ and $P_2 = b \to c$ is demonstrated in Fig. 3.

2.3 Parallel

The expression $P = \Vert\{P_1, \ldots, P_n\}$ denotes a process P, which executes all processes P_i in parallel. If processes have common events these events have to be synchronized. The operator $\Vert$ is an extension of Hoares full synchronous composition (FSC) (Hoare 1985), involving the explicit process synchronization operator $\&$.

Two processes, $P_6 = a \& c \to b$ and $P_7 = c \& e \to d$, have one common event c. These processes are to be executed in parallel (as shown in Fig. 4). The synchronized event $a\&c$ in process P_6 is required to occur at the same time as the synchronized event $c\&e$ in P_7.

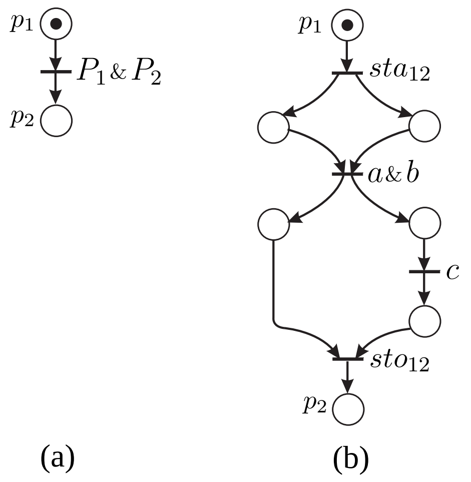


Figure 3: The process $P = P_1 \& P_2$ given in (a) and (b) as PPN models using the processes $P_1 = a$ and $P_2 = b \rightarrow c$.

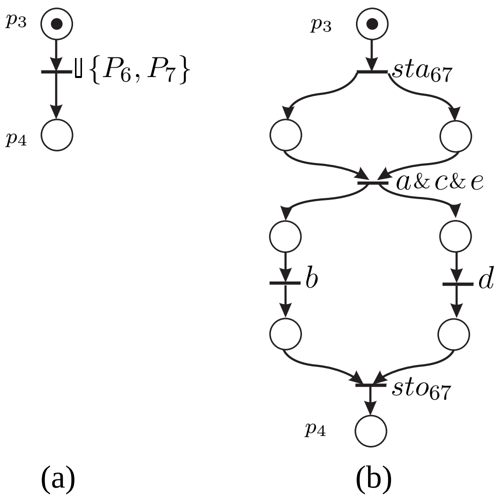


Figure 4: The process $P = \uplus\{P_6, P_7\}$ given in (a) as a PPN model and in (b) as a PN (including synchronized events).

3 Resources and routing specifications

In this section a formal description of the different building blocks in a resource allocation system is presented. These building blocks are a set of resource models (representing the plant), a set of routing specifications, and a supervisor which synchronizes the individual products utilization of the shared resources.

We assume in this paper the plant to consist of two generic classes of resources (equipment devices), namely *processors* (units) and *transporting devices*. *Processors* are typically tanks, reactors, and other container-like units. *Transporting devices*, on the other hand, have as their main task to open and close connections between processors causing and preventing material flow. Typical examples are valves and pumps. In this paper we exemplify processors with tanks and transporting devices with on/off valves.

3.1 Routing specification

Every product to be manipulated have its own route through a system. This is specified by a routing specification S_i that describes which operations an product are to undergo, in which order these operations are to be executed, and which resource(s) that may be used for each individual operation.

3.2 Resources

There are two different resources in our batch plant example, producers and transporters. Producers are either booked b_ℓ^i or unbooked u_ℓ^i, Fig. 5a, where the index ℓ and i refers to which resource it is and which routing specification using it. In the rest of this paper it will be assumed that only a single routing specification is using a specific resource which means that the index specifying this can be skipped.

Transporters, Fig. 5b, can either be booked b_ℓ^i or closed (blocked) c_ℓ^i as well as unbooked u_ℓ^i and unblocked r_ℓ^i. It can be blocked in order to guarantee that no other routing specification can use the particular resource at the same time. A transporter can be blocked by more than one routing specification (recipe) c_ℓ^j. Note that a blocked transporter can be booked by the same routing specification and vice versa.

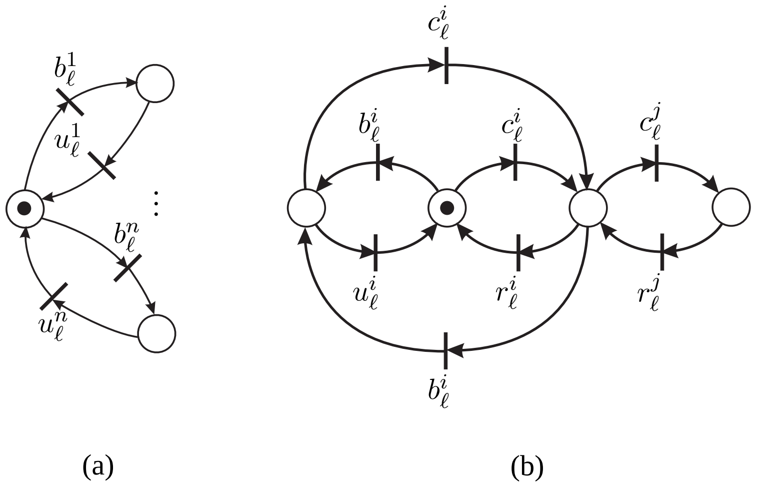


Figure 5: (a) Resource model of producers, and (b) resource model of transporters.

3.3 Supervisor

The purpose of the supervisor is to synchronize the products utilization of the common, available resources. It is important for this utilization to be as efficient as possible. Unnecessary restrictions must be avoided, and as much flexibility as possible be given to the system, without danger of running into blocking states or other forbidden configurations.

As a first step to obtain a supervisor, the parameterized resource models are transformed based on the current routing specifications. The transformed resource models are synchronized with the routing specifications. Assume that m specifications $Sp_1, \ldots, Sp_m$ are given which altogether use n resources $R_1, \ldots, R_n$. Then a complete model of the system is given by applying the parallel operator $\|$ on all resource and specification models.

Then the model

$$Sp = \Cup\{Sp_1, \ldots, Sp_m\} \tag{1}$$

is a first *specification* of the desired behavior of the plant

$$R = \Cup\{R_1, \ldots, R_n\} \tag{2}$$

Also note that the specifications S_i run independently of each other, since our modelling approach implies that the specification alphabets are disjunct.

The resulting model give what is called the *global specification*

$$S_0 = \Cup\{Sp_1, \ldots, Sp_m, R_1, \ldots, R_n\} \tag{3}$$

This is a first candidate for a possible supervisor S, and in fact it is a model of the controlled closed loop system. This model may however be blocking or noncomplete, (Ramadge and Wonham 1987). The first aspect has to do with liveness and the second one is related to uncontrollable events, which we do not consider in this paper.

The global specification S_0 therefore has to be manipulated to result in an appropriate supervisor. This is formally expressed by the operator $\mathcal{NB}$, which removes blocking states from S_0 to make it trim. The synthesized supervisor is now expressed as

$$S = \mathcal{NB}(S_0) = \mathcal{NB}(\Cup\{Sp_1, \ldots, Sp_m, R_1, \ldots, R_n\}) \tag{4}$$

4 The join and split operation

In this section we will introduce two additional building blocks that may be used to model material transfer in both manufacturing systems and batch processes. These building blocks are the split and join operation.

A process function

$$AJ(X, Y, Z) = X[Z] + Y[\overline{Z}] \tag{5}$$

with three variables, X, Y, and Z is introduced in order to simplify the specification of asynchronous material flows. The three variables are processes and the choice between process X or process Y is determined by the state of process Z, whether it is in its initial state or not.

4.1 Join operation

The join operation combines two or more material flows. One special case of the join operation is the synchronous join. This operation requires that all the joining parts are ready and all the required resources can be booked.

The other extreme is an asynchronous join. Taking the same interpretation as above, in the asynchronous case neither of the joining parts has to wait for the other in order to execute. The joining material flows act independently of each other and are synchronized after the joined material flows has finished. The asynchronous join is more complex to model, because all the material flows that are to be joined together can not book the same resources. Therefore it is necessary that the model restricts the booking of the required resources so that each resource is only booked once.

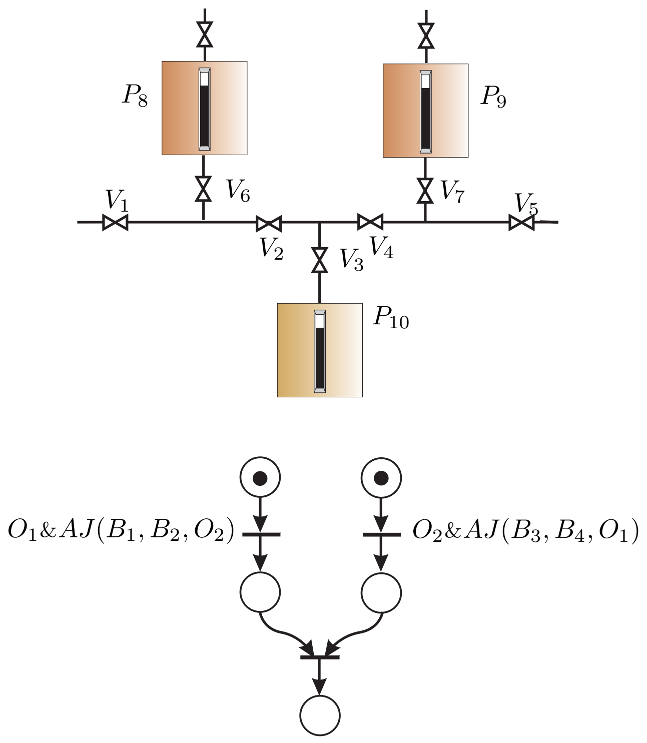


Figure 6: Asynchronous join given as a PPN model with booking processes given in Table III.2. Note that, a join operation involves only one recipe S and S can therefore first block a valve and then book it, see Fig. 5.

In Fig. 6 a PPN model with two operations O_1 and O_2 is given. The required resources for each operation is listed in Table III.2. A resource can only be booked once and it is therefore necessary to specify so that this is guarantied. It is done by using two booking processes for each operation cf. Table III.2. The operation that start its execution first book all the required resources for that operation. When the second operation starts its execution it only books the additional resources that are not already booked. This is controlled using the process function in (5).

Note that apart from the valves and tanks that are booked, it is necessary for the recipe to close (block) some valves in order to secure the path through the plant. Parenthesis around the required resource in Table III.2 specifies that it need to be blocked. The blocking events is denoted c_i.

Table III.2: Required resources for operation O_1 and O_2 for asynchronous join.

Operations	Required resources
O_1	$P_8, P_{10}, (V_2), V_3, V_4, (V_5), V_7$
O_2	$P_9, P_{10}, (V_3), V_4, V_5, (V_6), V_8$
Booking processes	
$B_1 = b_8 \,\&\, b_{10} \,\&\, c_1 \,\&\, b_2 \,\&\, b_3 \,\&\, c_4 \,\&\, b_6$	
$B_2 = b_8 \,\&\, c_1 \,\&\, c_4 \,\&\, b_6$	
$B_3 = b_9 \,\&\, b_{10} \,\&\, c_2 \,\&\, b_3 \,\&\, b_4 \,\&\, c_5 \,\&\, b_7$	
$B_4 = b_9 \,\&\, c_2 \,\&\, c_5 \,\&\, b_7$	

4.2 Split operation

The split operation separates two or more material flows. In the same way as for the join operation there are two special cases for the split operation. The first is the synchronous split operation, which requires that all the involved parts are ready and all the required resources can be booked or blocked.

The other extreme is an asynchronous split. Taking the same interpretation as above, in the synchronous case neither of the involved parts has to wait for the other in order to execute. The separated material flows act independently of each other and are synchronized after the separated material flows has finished. In the same way as for the asynchronous join it is necessary to control so that only the first of the material flows books the common resources.

In Fig. 7 tank P_9 and P_{10} are to be filled with the material from a third tank P_8. If it is not required that both tank P_9 and P_{10} are filled at the same time it can be modelled as an asynchronous split. If for instance tank P_8 and P_9 are ready together with the necessary valves but tank P_{10} is occupied then tank P_8 and P_9 can be booked together with the involved valves in order to start filling tank P_9. When tank P_{10} becomes ready it is booked together with the valves that are needed except for those valves that are already booked.

In Fig. 7 the PPN model for the asynchronous split operation is presented together with the two operations O_1 and O_2. The required resources for each operation are listed in Table III.3.

5 Tank example

Two products with recipes S_1 and S_2 will be produced in the plant given in Fig. 12. Recipe S_1 describes two alternative paths throw the plant. The raw materials M_1 and M_2 in T_{28} and T_{29} respectively, are first combined into either reactor P_{19} or P_{21}. This combination needs to be synchronized in order to keep the concentration level of the mixture kept between certain boundaries. After this the content of P_{19} or P_{21} is emptied into the adjacent filter P_{22} and P_{23} respectively. The content is then separated into T_{28} and P_{24} or P_{27}. Recipe S_2 combines raw materials M_2 and M_3 into reactor P_{20}. M_2 and M_3 can be filled into the reactor

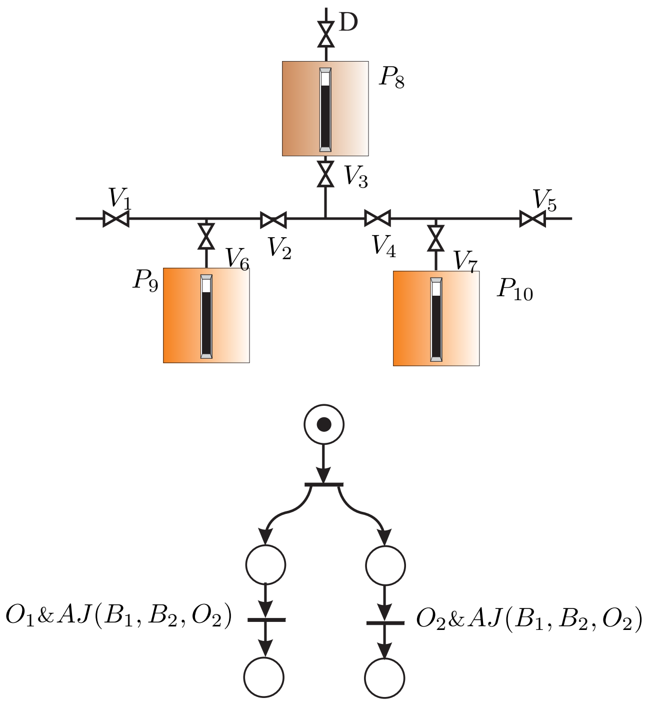


Figure 7: Asynchronous split given as a PPN model with booking models given in Table III.3.

independently. The content in P_{20} is then dumped into either storage tank P_{25} or P_{26}.

Table III.4 and Table III.5 show all the required resources for each operation in S_1 and S_2 respectively. Observe that supply tanks do not need to be booked or unbooked. The reactors have the simple resource model with only two states for each routing specification as in Fig. 5a. The valves are modelled as in Fig. 5b and can be both booked and blocked from the initial state. It is required to create booking and unbooking processes that explicitly declare if a valve is to be booked or blocked for a specific operation. Note that the booking and unbooking processes are automatically generated from the list of required resources, B_i and U_i cf. Table III.2.

Recipe S_1 is in Fig. 8 modelled with six operations $O_1 \ldots O_6$ one for each material

Table III.3: Required resources for operation O_1 and O_2 for asynchronous split.

Operations	Required resources
O_1	$P_8, P_9, (V_2), V_3, V_4, (V_5), V_7$
O_2	$P_8, P_{10}, (V_3), V_4, V_5, (V_6), V_8$
Booking processes	
$B_1 = b_8 \& b_9 \& c_1 \& b_2 \& b_3 \& c_4 \& b_6$	
$B_2 = b_9 \& c_1 \& b_2 \& b_6 \& c_4$	
$B_3 = b_8 \& b_{10} \& c_2 \& b_3 \& b_4 \& c_5 \& b_7$	
$B_4 = b_{10} \& c_2 \& b_4 \& c_5 \& b_7$	

transfer. The merging of material from the supply tanks in this recipe is synchronous since it requires that both material flows are available before any transfer can occur. The first choice creates two separate paths in the batch plant and it is natural to model this as an alternative between two sequences of operations, $O_1 \rightarrow O_3 \rightarrow O_5$ and $O_2 \rightarrow O_4 \rightarrow O_6$. Every transition in the PPN is modelled with the specific operation together with booking processes for all required resources. Each transition expression involve the booking of the resources required for the particular operation as well as the unbooking of the resources when the operation has completed its execution. In that a few resources are required for the following operations and are not included in the unbooking process.

Table III.4: Required resources for each operation, O_1 - O_6 given for routing specification S_1. Resources within bars are only required to be blocked cf. Fig. 5b.

Operation	Required resources
O_1	$P_1, V_1, V_2, V_4, V_7, (V_5), (V_{10})$
O_2	$P_3, V_1, V_2, V_4, V_5, V_6, V_9, (V_3), (V_7), (V_8), (V_{12})$
O_3	$P_1, P_4, V_{10}, (V_{13}, (V_{15}))$
O_4	$P_3, P_5, V_12, (V_{14}), (V_{18})$
O_5	$P_4, P_6, V_{13}, V_{15}, (V_{14})$
O_6	$P_5, P_8, V_{14}, V_{18}, (V_{13})$
Booking process	
$B_1 = b_{19} \& b_1 \& b_2 \& b_4 \& b_7 \& c_5 \& c_{10}$	
$B_2 = b_{21} \& b_1 \& b_2 \& b_4 \& b_5 \& b_6 \& b_9 \& c_3 \& c_7 \& c_8 \& c_{12}$	
$B_3 = b_{22} \& b_{10} \& c_{13} \& c_{15}$	
$B_4 = b_{23} \& b_{12}, c_{14}, c_{18}$	
$B_5 = b_{24} \& b_{13} \& b_{15} \& c_{14}$	
$B_6 = b_{26} \& b_{14} \& b_{18} \& c_{13}$	
Unbooking process	
$U_1 = u_1 \& u_2 \& u_4 \& u_7 \& r_5 \& r_{10}$	
$U_2 = u_1 \& u_2 \& u_4 \& u_5 \& u_6 \& u_9 \& r_3 \& r_7 \& r_8 \& r_{12}$	
$U_3 = u_{19} \& u_{10} \& r_{13} \& r_{15}$	
$U_4 = u_{21} \& u_{12}, r_{14}, r_{18}$	
$U_5 = u_{22} \& u_{24} \& u_{13} \& u_{15} \& r_{14}$	
$U_6 = u_{23} \& u_{26} \& u_{14} \& u_{18} \& r_{13}$	

Recipe S_2 in Fig. 9 is modelled with three operations O_1, O_2, and O_3. Operations O_1

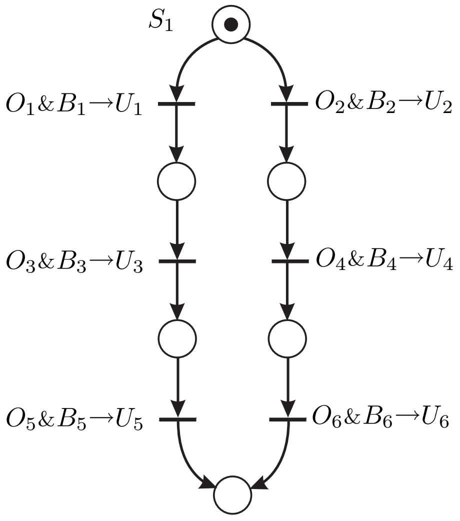


Figure 8: Recipe S_1 given as a PPN model with booking/unbooking processes cf. Table III.2.

and O_2 can be executed in parallel since M_2 and M_3 can be transferred independently. This results in an asynchronous join and places high demands on how the resources are booked, which is described in Section 4. The last operation O_3 dumps the material in either of two storage tanks. Due to the asynchronous join in this specification it is necessary to use two booking and two unbooking processes for each operation, cf. Section 4.1. Observe that the unboking of the resources is also restricted in Fig. 9. This is important because it would otherwise be possible to unbook resources that are still required by another operation. In Fig. 9 every transition in the PPN is modelled with the specific operation, O_1, O_2, and O_3 together with booking processes for all required resources.

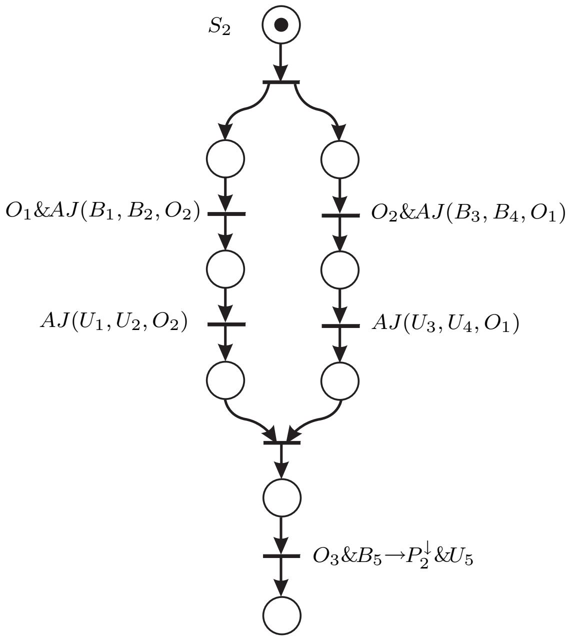


Figure 9: Recipe S_2 given as a PPN model using process functions for asynchronous join and with booking/unbooking processes cf. Table III.3.

Table III.5: Required resources for each operation given for routing specification S_2. Resources within bars are only required to be blocked cf. Fig. 5b.

Operation	Required resources
O_1	$P_2, V_2, V_5, V_8, (V_4), (V_6), (V_{11})$
O_2	$P_2, V_3, V_6, V_8, (V_5), (V_9), (V_{11})$
O_2	$P_2, P_7, V_{11}, V_{16}, (V_{17})$ or $P_2, P_8, V_{11}, V_{17}, (V_{16})$
Booking processes	
$B_1 = b_{20} \& b_2 \& b_5 \& b_8 \& c_4 \& c_6 \& c_{11}$	
$B_2 = b_2 \& b_5 \& c_4$	
$B_3 = b_{20} \& b_3 \& b_6 \& b_8 \& c_5 \& c_9 \& c_{11}$	
$B_4 = b_3 \& b_6 \& c_9$	
$B_5 = b_{25} \& b_{11} \& b_{16} \& c_{17} + b_{26} \& b_{11} \& b_{17} \& c_{16}$	
Unbooking processes	
$U_1 = u_{20} \& u_2 \& u_5 \& u_8 \& r_4 \& r_6 \& r_{11}$	
$U_2 = u_2 \& u_5 \& r_4$	
$U_3 = u_{20} \& u_3 \& u_6 \& u_8 \& r_5 \& r_9 \& r_{11}$	
$U_4 = u_3 \& u_6 \& r_9$	
$U_5 = u_{25} \& u_{11} \& u_{16} \& r_{17} + u_{26} \& u_{11} \& u_{17} \& r_{16}$	

6 Conclusion

Specifications of resource allocation systems tend to become very complex for large systems. The present work suggested a way of dealing with specification of complex systems in a more efficient manner, by the use of a specification formalism called PPN. By combining the graphical advantages of PNs with the compactness of process algebra we achieve a formalism, which is able to deliver specifications of resource allocation systems that are both concise and easily interpreted.

More specifically, the present paper has specified high level routing specifications for products to be produced in a chemical batch plant. It is also made evident how parts of a specification may be more ideally suited for either PN or process algebra expressions. Based on the created specifications a supervisor may also be synthesized, which synchronizes the utilization of available resources.

The language is illustrated for different combinations of multiple and alternative resource allocation systems, especially for a batch process problem that follows throughout the paper, where process operators such as synchronization and alternative and restriction expressions are shown to be very useful.

References

Åkesson, K. (2002). *Methods and tools in supervisory control theory*, Phd thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 431.

Arnold, A. (1994). *Finite Transition Systems: Semantics of Communicating Systems*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Basten, T. (1998). *In Terms of Nets:System Design with Petri Nets and Process Algebra*, PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Best, E., Devillers, R. and Koutny, M. (1998). Petri nets, process algebras and concurent programming languages, *Proc of ICM'98*, Berlin, Germany.

Best, E., Devillers, R. and Koutny, M. (2002). The box algebra = petri nets + process expressions, *Information and Computation* (178): 44–100.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, Kluwer Academic Publishers.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P. and Lennartson, B. (2005a). A high level specification language based on process algebra and petri nets, *To be submitted to Transactions on Automation Science and Engineering* .

Falkman, P. and Lennartson, B. (2005b). Using a high level language for verification and control synthesis of discrete event systems, *Submitted to Transaction on Control System Technology* .

Falkman, P., Lennartson, B. and Tittus, M. (2001). Modeling and specification of discrete event systems using combined process algebra, *Proc. of 2001 IEEE/ASME Advanced Intelligent Mecatronics*, COMO, Italy.

Falkman, P., Nielsen, J. and Lennartson, B. (2004). A method for automated generation of discrete event systems from step ap214 for process planning and control, *Submitted to Journal of Manufacturing Systems* .

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Hopcroft, J., Motwani, R. and Ullman, J. (2001). *Introduction to Automata Theory, Languages and Computation*, 2nd ed. edn, Addison-Wesley Series in Computer Science, Addison-Wesley.

Mayr, R. (1997). Combining petri nets and pa-processes, *Theoretical Aspects of Computer Software (TACS'97), volume 1281 of Lecture Notes in Computer Science*, Sendai, Japan.

Olderog, E.-R. (1987). Petri nets and algebraic calculi of processes, *Advances in Petri Nets, 266 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany* pp. 196–223.

Pena, M. and Cortadella, J. (1996). Combining process algebras and petri nets for the specification and systethis of asynchronious circuits, *Proc of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Fucushima, Japan.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes, *SIAM J. Control Optim.* **25**(1): 206–230.

Rondogiannis, P. and Cheng, M. (1994). Petri-net-based analysis of process algebra programs, *Elsevier Science Publisher B.P., Science of Computer Programming* pp. 55–89.

generation of ppns for control purposes

In: *Journal of Systemics, Cybernetics and Informatics*, Vol 1, Number 5. :

Automatic Generation of Object Models for Process Planning and Control Purposes using an International standard for Information Exchange

P. Falkman[†] and J. Nielsen[*] and B. Lennartson[†]
[†]Control and Automation Laboratory
Chalmers University of Technology

[*]Department of Precision Machinery Engineering
The University of Tokyo Hongo 7-3-1

In this paper a formal mapping between static information models and dynamic models is presented. The static information models are given according to an international standard for product, process and resource information exchange, (ISO 10303-214). The dynamic models are described as Discrete Event Systems. The product, process and resource information is automatically converted into product routes and used for simulation, controller synthesis and verification. A high level language, combining Petri nets and process algebra, is presented and used for specification of desired routes. A main implication of the presented method is that it enables the reuse of process information when creating dynamic models for process control. This method also enables simulation and verification to be conducted early in the development chain.

1 Introduction

In order to be competitive, engineering companies of today have to be flexible and responsive to rapidly changing market needs. For this reason, it is important for companies to decrease the time to market while still maintaining or increasing product quality, all at a low cost. A step towards decreasing the time to market is a more efficient information exchange between product and manufacturing systems design.

Making the information exchange more efficient means that information about product design solutions becomes instantly available to engineers involved in the manufacturing system design. More concretely a process planner will start the documentation of how to manufacture a product based on preliminary design solutions already during the product design. If information can be made instantly available to engineers, the iteration cycle between product and manufacturing systems design can also be made shorter.

This process documentation will be used as a base for simulating how the introduction of a new product will affect an existing, or new, manufacturing system. The outcome of the simulation will influence the final design solution of the product as well as the manufacturing system. The created dynamic models will, in addition to this, also be used for verification and automatic controller synthesis.

This paper focuses on verification and automatic controller synthesis. The controller synthesis includes two levels of control descriptions. First, the resource allocation system and second, a more detailed control of specific applications, e.g. control of a robot cell. In the resource allocation system a number of products utilize a number of shared resources which are to be booked and unbooked. A high level language intended to simplify the specification of desired routes is presented here. This modelling language combines Petri nets and process algebra in order to achieve compact representations of the product routes. The more detailed control descriptions involves specific control for each resource and can be seen as a decomposition from the higher level resource allocation system which is not dealt with in this paper. The focus in this paper is, however, on the resource allocation system.

The process plan defines process information as a set of product, process and resource characteristics, defining what to produce, and how it should be done. This information can be created using several different systems, such as CAD-systems, Robot simulation and Off-Line Programming (OLP) systems.

In order to automatically generate dynamic process models for process control purposes, a mapping is necessary. This mapping should define the relationship between the static information and the dynamic process models.

Much research has already been conducted on information and discrete event modelling, e.g. (Schenk and Wilson 1994), (Scheller 1990), and (Eversheim et al. 1991) discussing information modelling and (Cassandras and Lafortune 1999), (Hoare 1985), and (Ramadge and Wonham 1987) discussing discrete event modeling. However, little has been investigated concerning the connection between information and dynamic models, i.e. how an information structure could be mapped to the dynamic structure of a process plan. The purpose of this paper therefore is threefold: first to capture the requirements of the information structure, second to capture the requirements of the dynamic structure, and third to show how the mapping between the information and dynamic structure could be realized.

The information structure is given according to the ISO10303-214 or the STEP-standard (STandard for Exchange of Product model data). The mapping has been achieved by analyzing the information structure, (ISO 10303-214), and the dynamic structure, (the MPPN-model) which was introduced in (Falkman and Lennartson 2001), to gain knowledge of the semantics of their respective objects and structures. The gained knowledge has then been synthesized to result in the semi-formally defined mapping model. Finally, the result has been validated using a case study at Scania Oskarshamn, Sweden. This has been done by populating the ISO 10303-214 model with data from the Scania case, and then implementing the mapping method in order to automatically create an MPPN-model based on the Scania data. The case is presented below and developed throughout the paper.

Example 13 – A robot cell in a Scania factory The robot cell shown in Figure 1 consists of six resources, a robot R, a gripper G, a welding machine W, two output buffers B_1 and B_2, an input buffer I, an operator O, and two fixtures left FL and right FR.

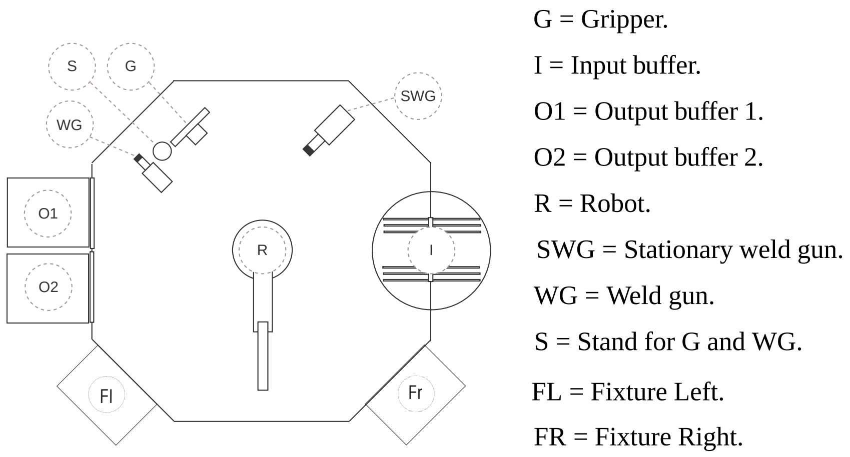


Figure 1: A manufacturing cell.

The Scania robot cell involves two main processes, **StationaryWelding** and **RobotWelding**.

StationaryWelding As input to the robot cell there are geometrically welded plates placed on the turn table. StationaryWelding involves three sub-processes: **Get**, robot and gripper is used to get workpiece from instation. **Weld**, robot, gripper, and stationary weld gun is used to weld approximately 30 weld spots. **Put**, robot and gripper is used to put the workpiece in outstation $O1$ or $O2$.

RobotWelding As input to the robot cell an operator places the geometrically welded plates in one of two fixtures. RobotWelding also involves three sub-processes: The first is **Place** and involves an operator placing a workpiece on either the right or the left fixture. The second is **Weld** where the robot, the weld gun, and one of the fixtures LF or RF is used to weld about 30 weld spots and finally the third and final process **Put** where the robot together with the gripper is used for putting the workpiece in outstation $O1$ or $O2$.

$\square$

In the following two sections an introduction to both the MPPN modelling language and the ISO 10303-214 standard is presented. In addition to this a comparison between the static and dynamic models is made with respect to the product, process, and manufacturing resource (PPR) representation in both MPPN and ISO 10303-214 (AP214).

2 Mixed Process algebra Petri Net

The MPPN language combines Petri nets and process algebra in order to create product specifications. The MPPN language uses process operators for alternative, synchronization in order to realize compact specifications.

Process operators

The transition between two Petri net places in an MPPN is a process P. A process $P = a_1 \rightarrow P_1$ describes that, first the event a_1 occurs, then it behaves like a process P_1.

Alternative The alternative operator $+$ specifies that there is a choice between two processes. Let two processes be defined as $P = a_1 \rightarrow P_1$ and $Q = b_1 \rightarrow Q_1$. Then an alternative between these two processes is described as

$$P + Q = a_1 \rightarrow P_1 \mid b_1 \rightarrow Q_1 \tag{1}$$

using Hoare's (Hoare 1985) choice symbol $|$. This implies that either event a_1 occurs followed by process P_1 or event b_1 occurs followed by process Q_1.

Synchronization The nonstandard synchronization operator $\&$ implies that one or more processes are to be synchronized, with no respect to common events, and executed in parallel. Similar ideas for event synchronization can be found in (Arnold 1994). Again, consider two processes $P = a_1 \rightarrow P_1$ and $Q = b_1 \rightarrow Q_1$. The synchronization operator & can be described as

$$P\&Q = a_1\&b_1 \rightarrow P_1\&Q_1 \tag{2}$$

This means that a_1 in P occurs at the same time as b_1 in Q. This synchronized event is denoted $a_1\&b_1$. The synchronization operator $\&$ is useful when *flexibility* and *reusability* is desired.
[1]

Parallel processes Parallel processes are defined using the Petri net constructs instead of introducing a parallel process operator. This is done in order to preserve a good graphical presentation of the modelled system. In Figure 2 there are two processes P and Q which are to be executed in parallel.

Product Model

The dynamic model of a product is the process model that will be described in the next section. However, the static information for a product, e.g. product id, may, in the MPPN model, be assigned to a token. The number of tokens control the number of products manufactured and the number of products or product parts that are allowed in to the manufacturing system at the same time. Note that this may involve colored Petri nets but this extension is not emphasized in this paper.

[1]The synchronization operator is redefined in the PPN language (Falkman and Lennartson 2005a).

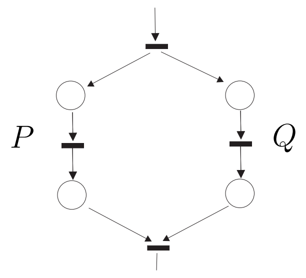


Figure 2: Two processes P and Q are executed in parallel.

Process Model

The *resource allocation system* involves a set of products that share a set of resources within a manufacturing system. To ensure that only one product at a time is using a specific resource it is necessary for each resource to be *booked* by a specific product. It is also important to control that there are no *blocking* or *deadlock states*. A routing specification specifies a products route through a resource system and may be described on two levels:

- a *high level routing specification* (HRS) that describes which processes an object are to undergo, in which order these processes are to be executed, and which resource(s) that may be used for each individual process.
- a *booking and unbooking specification*, which describes on a more detailed level how the shared resources are to be booked and unbooked, based on the HRS, to obtain the desired route through the resource system.

Process Operation (PO) In the resource allocation system the transition between two places in the HRS is a *process operation* (PO). A PO involves two processes, a booking process B and an unbooking process U. A booking and unbooking model is automatically created given an HRS.

Example 14 – Robot cell An HRS to the left in Figure 3 specifies the three processes involved for the StationaryWelding, in the robot cell example. The first PO requires three resources: the robot R, the gripper G and the In-station I. The second resource requires the robot R, the gripper G and the weld gun WG. To the right in Figure 3 is described on a more detailed level how the resources involved are to be booked and unbooked. Note that resources that are required in several operations are not unbooked.

$\square$

Manufacturing System Model

A model of the manufacturing system, for the resource booking system, is created by synchronizing all of the involved resource models.

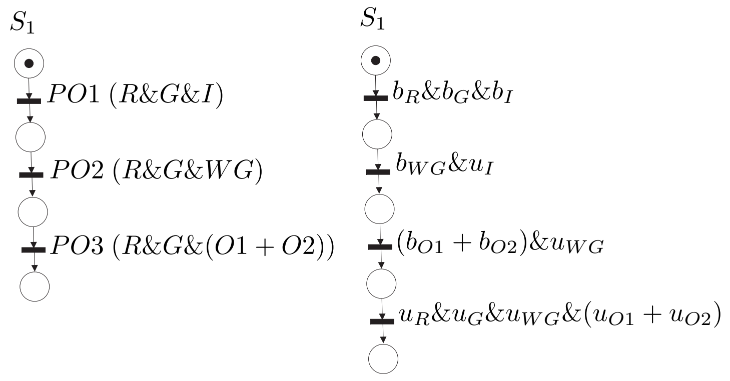


Figure 3: A routing specification is given as an HRS to the left and a booking and unbooking specification to the right. Note that resources that are required in more than one operation in a row are not unbooked.

3 The STEP AP214 Model

In this section the product, process, and manufacturing resource (PPR) representation in ISO 10303-214 (AP214) will be described. The objective is to describe where to find the information needed to generate the MPPN-model. Before the PPR-model is described a brief introduction to ISO 10303 (STEP) and the EXPRESS language will be given.

ISO 10303 STEP is an international standard that “provides a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged” (TC184/SC4 1994). The term exchange should here be interpreted as the exchange of data between computer systems in environments associated with the complete life-cycle of a product, including manufacturing.

The standard consists of different parts, called application protocols, that define the scope, context, and information requirements for a particular application, e.g. the automotive industry (AP214), or electrical design and installation (AP212).

EXPRESS language The EXPRESS language is a formally specified and structured language (Schenck and Wilson 1994) used to define the application interpreted models in STEP. The EXPRESS is an earlier and more general alternative to UML. The basic constructs of EXPRESS or EXPRESS-G (a graphical subset of EXPRESS) is the entity and the attribute. An entity is similar to an object in object-oriented programming.

Graphically, in EXPRESS-G, an entity is represented as a box with a name written in it. Attributes are represented by a line ending with a small circle, showing the direction of the relationship. They are labelled with the name of the attribute as well as any cardinality constraints. A dashed line represents an optional attribute, whereas a thick line represents a supertype-subtype relationship, i.e. the same as inheritance in object-oriented programming.

Product Model

The most important product entities in AP214 is the *item* (i) and *item_version* (iv). These are the holders of product meta-data, such as identifiers, version data, classification data and much more.

Process Model

The process model in AP214, cf. Figure 4, has a central role in the generation of the MPPN-model. It is the holder of all the necessary process information, such as plan identifier, relationships between processes etc.

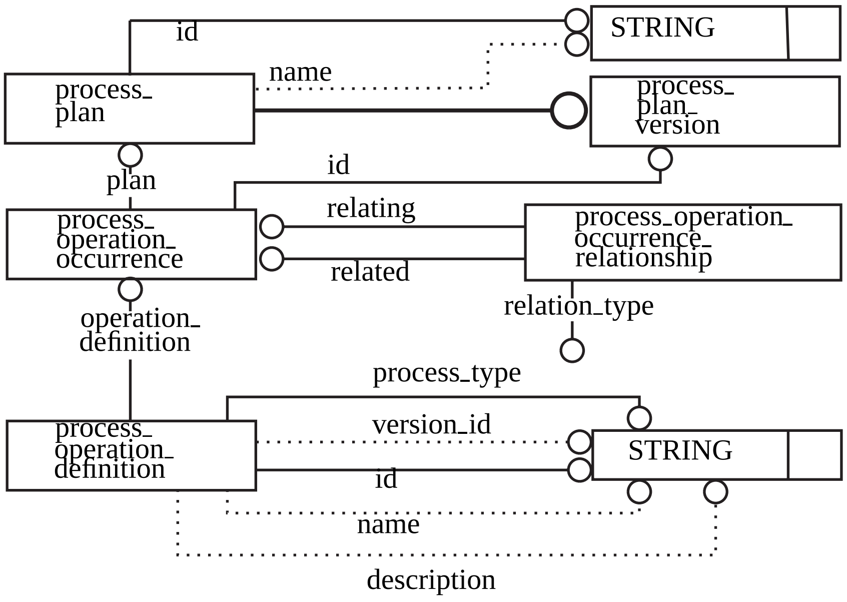


Figure 4: Representation of process data in AP214.

The process model consists of a structure to hold meta-data about a process plan. This structure is identified by the *process_plan* (pp) and *process_plan_version* (ppv) in Figure 4. A *process_plan*, consists of one or more processes represented by the *process_operation_occurrence* (poo). The *process_operation_occurrence* represents the occurrence of a process in a process plan. More specifically, it represents the occurrence of a definition of a process, the *process_operation_definition* (pod). This mechanism enables the reuse of a definition in several different places in a process plan as well as in several different process plans and thus, different versions of a plan can reuse definitions that have not been changed from a former version. For instance, alternative resource for the same operation would be represented by the same definition but with different resources assigned to different *process_operation_occurrence*s all representing the same definition.

The structure of the process plan is represented on the *process_operation_occurrence* level, i.e. the level where sequence, alternative, simultaneity, and substitution relationships between processes are represented. This relationship is represented by the *process_operation_occurrence_relationship* (poor) where the attribute *relation_type* holds the type of relationship. The attribute *relating* points in the direction of the *process_operation_occurrence* prior to the *process_operation_occurrence* pointed out by the attribute *related*.

Manufacturing System Model

Manufacturing resources can be represented in several different ways in AP214, depending on the level of detail and the design life cycle stage.

The *single_instance* and *physical_instance* are both instances of an abstract representation of a manufacturing resource (*item*), but there is one significant difference. The *single_instance* represents an occurrence of a type of manufacturing resource whereas the *physical_instance* represents a physical resource on the shop floor. Thus the *single_instance* is better used for planning purposes before a physical resource exists and the *physical_instance* is better used when there already exists a physical resource.

4 Mapping of AP214 into MPPN

In this section a description of the relationship between AP214 and the MPPN product, process and manufacturing system models is given. Examples are given in order to illustrate the mapping of the static description of the product to be manufactured into a dynamic model using the graphical notation of both descriptions. The actual algorithm is not shown mainly due to lack of space. A simplified algorithm is given in (Falkman, Nielsen and Lennartson 2002).

Product

For the purpose of creating a product in the MPPN-model only a product identifier is needed. The product identifier is represented by the *item_version.id* in AP214. This identifier will be assigned to the *token* in MPPN which implies a colored extension of the MPPN. The product identifier is related to process information via the *process_plan.produced_output*.

Manufacturing system

The attributes of the manufacturing resources that are needed in order to generate a MPPN-model are the *single_instance.id* and *physical_ instance.serial_number*.

Process

Process information needed in order to create an MPPN-model is *process_plan.id, process _operation_occurrence.id*, *process_operation_definition.id*, and information about relationships between processes. Table IV.1 describes the use of AP214 process model information when creating an MPPN-model.

AP214 Information	**MPPN-model**
Plan (pp) identifier	Routing specification identifier
Occurrence (poo) identifier	Petri net place (pnp) identifier
Definition (pod) identifier	Process operator (PO) identifier
Relationship (poor) type	Net structure

Table IV.1: The use of AP214 information when creating a MPPN-model.

The differences between *product_operation_definition* and *product_operation_occurrence* are several but the most important one in this paper is that the *product_operation_definition* gives a general description of what an operation involves, whereas the *product_operation_occurence* describes on a more detailed level which resource to use in a specific operation. For instance, a *product_operation_definition* may be executed by two different *product_operation_occurences* in that that they use different resources. In the MPPN model a process that only differs in which resource they require is regarded as the same *process operation*. This means that it is natural to use the *product_operation_ definition* identification when translating into MPPNs.

Alternative resources

In an earlier phase of this project alternative resources has already been implemented. In the STEP standard alternative resource is represented by *process_operation_occurences* that refers to the same *process_operation_definition* and also refers to each other with the *process_operation_occurence_relationship* attribute *relation_type* equal to 'alternative', cf Figure 4. In the MPPN the alternative resource is represented by a number of alternative resources in the transition equation, cf 5.

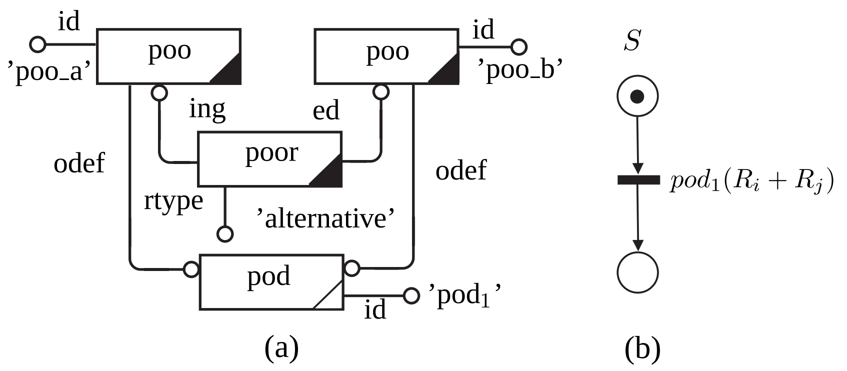


Figure 5: Alternative process given as a High Level Routing specification (b) and an instantiated STEP AP214-model (a). This is alternative processes in the STEP standard with *process_operation_occurences* refereing to the same *process_operation_definition*. Each *process_operation_occurence* relates to one resource each R_1 and R_2

In figure 5(a) a small part of a populated STEP model is represented. There are two *process_operation_occurences*, with respective id 'poo_a' and 'poo_b', which refers to each other with the *relation_type* attribute 'alternative'. Both of the *process_operation_occurences* also refer to the same *process_operation_definition* pod_1. This implies that the same operation is being executed by both *process_operation_occurences*, however these two *process_operation_occurences* are referring to different resources which is not shown in Figure 5(a). In Figure 5(b) an HRS is described showing the use of the + operator for representation of the resource choice.

Alternative processes

Alternative, or split, differs from the earlier mentioned alternative resource in that it in STEP are *process_operation_occurences* that refer to each other with the *process_operation_ oc-*

curence_relationship with the attribute *relation_type* equal to 'alternative', cf Figure 4, but the different *process_operation_occurences* do not refer to the same *process_operation_definition*. This results in a split of the sequence in the MPPN in to two or more separate branches, cf Figure 6.

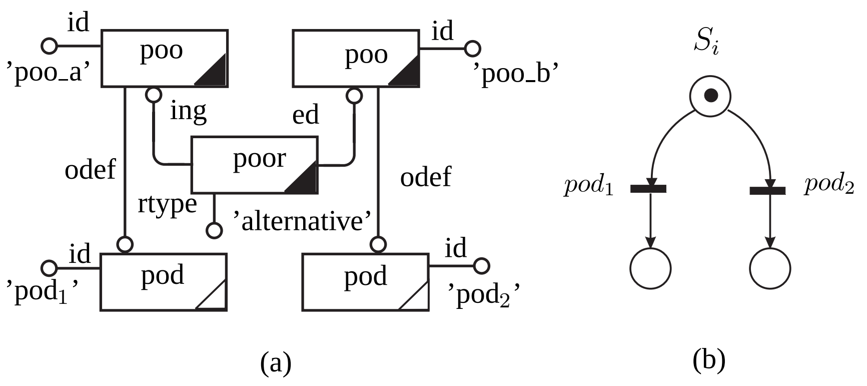


Figure 6: Alternative process given as a High Level Routing specification (b) and an instantiated STEP AP214-model (a). Note that this is different from alternative resources which in STEP is described by the fact that the involved processes are refereing to the same definition.

Parallel

Parallel processes are described in the MPPN in the same manner as in ordinary Petri Nets, c.f. Figure 2. In step this is modelled as two or more *process_operation_occurences* that refer to each other with the *process_operation_occurence_relationship* with the attribute *relation_type* equal to 'parallel', cf Figure 4.

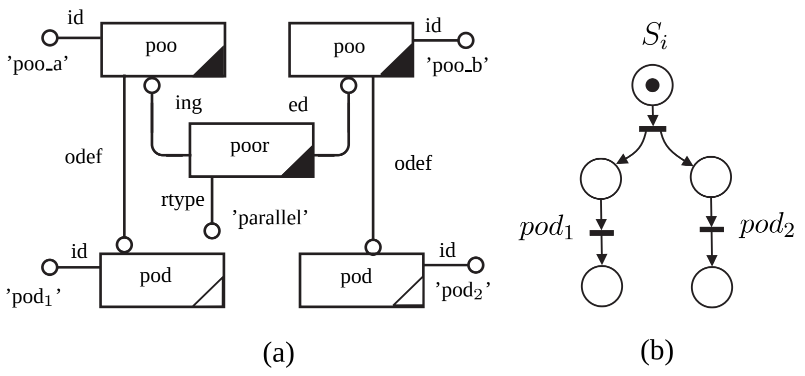


Figure 7: Parallel process given as a High Level Routing specification (b) and an instantiated STEP AP214-model (a).

Example 15 – Robot cell In Figure 8 shows how the *StationaryWleding* process in the Scania robot cell may be modelled with STEP. The same process is in Figure 9 described using the High level part of the MPPN language. As can be seen in Figure 8 there are four *process_operation_occurences* and only three *process_operation_definitions*. Two *process_operation_occurences* are referring to each other with the relation type 'alternative'.

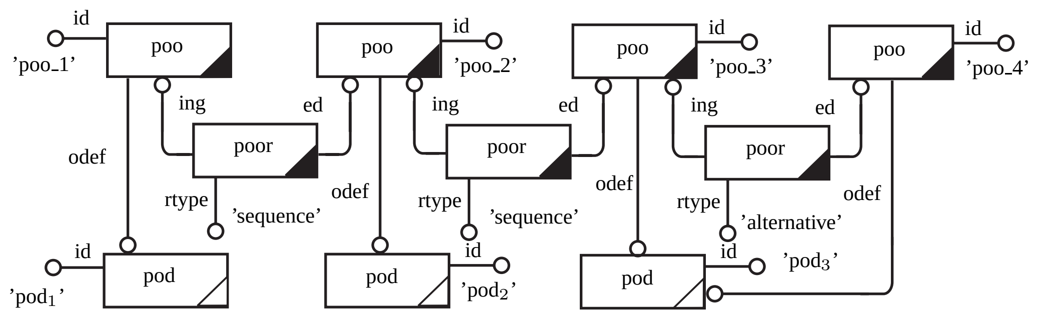


Figure 8: Instantiated example of the *StationaryWelding* process of the Scania robot cell given as a simplified instantiated STEP AP214 model. Note that the each *process_operation_occurence* relates to one or more resources but for clarity this is not shown in this picture.

Transferring the STEP model in Figure 8 into the MPPN model in Figure 9results in an MPPN with three operations executed in sequence. The three operations required a set of resources each and as a consequence of the two *process_operation_occurences* referring to the same *process_operation_definitions* exists alternative resources for $pod3$.

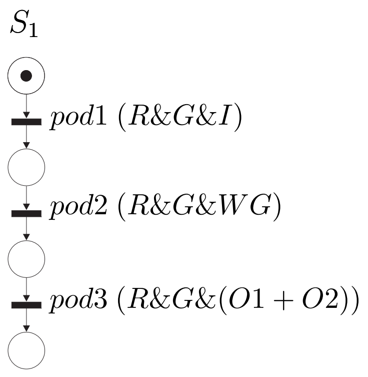


Figure 9: A routing specification given as an HRS and describing the *StationaryWelding* process of the Scania Robot cell.

□

The presented ideas so far involve only part of all the information needed in order to control a single cell or a whole plant. There is security information and much more but at this point we have concentrated on translating flow control information.

5 Conclusions and Future work

The suggested method implies a reliable framework for the exchange of control related information, which involves resource, product and process information. In addition to this it delivers the expected information fast, which is crucial when short lead times are required.

One of the main advantages of this method is that it involves, and uses, a well accepted international standard, STEP AP214, for the exchange of product, process, and resource related information.

In future work the entire method will be implemented and applied on a large industry case.

References

Arnold, A. (1994). *Finite Transition Systems: Semantics of Communicating Systems*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, Kluwer Academic Publishers.

Eversheim, W., Marczinski, G. and Cremer, R. (1991). Structured modelling of manufacturing processes as nc-data preparation, In Annals of the CIRP, volume 40/1.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P., Nielsen, J. and Lennartson, B. (2002). A formal mapping of static information models into dynamic models for process planning and control purposes, *Proc. of WODES 2002*, Spain.

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes, *SIAM J. Control Optim.* **25**(1): 206–230.

Scheller, A. (1990). Information modeling for distributed applications, *Proc of second IEEE Workshop on Future Trends of Distributed Computing Systems*.

Schenck, D. and Wilson, P. (1994). *Information Modeling: The EXPRESS Way*, Oxford University Press. ISBN: 0-19-508714-3.

Schenk, D. and Wilson, P. (1994). *Information Modeling: The EXPRESS Way*, ISBN 0-19-508714-3, Oxford University Press.

TC184/SC4, I. (1994). Iso 10303-1: Industrial automation systems and integration - product data representation - and exchange - part 1: Overview and fundamental principles, ISO Standard.

relationship between step and ppn models

Submitted to: *IEEE Transactions on Automation Science and Engineering*:

Automated Generation of STEP AP214 models from Discrete Event Systems for Process Planning and Control

P. Falkman[†] and J. Nielsen[*] and
B. Lennartson[†] and A. von Euler-Chelpin[‡]
[†]Signals and Systems
Chalmers University of Technology
[*]department of Precision Machinery Engineering
The University of Tokyo
[‡]Industriell Produktion, Kungliga tekniska högskolan

The aim of this paper is to show how the international standard STEP-AP214 can be used for communication and storing of process specifications. Even though there are several software tools available for the generation of both product- and resource information systems, there is still a lack of tools related to the STEP standard for producing process information, e.g. sequence of operations and system capabilities for resource allocation. Therefore such a tool is suggested, which makes use of a high level language for discrete event systems (DESs) based on process algebra and Petri nets. This language, called process algebra Petri net (PPN), has been developed in accordance with the process relations defined in STEP-AP214. It is specifically shown how process specifications created with the PPN tool can be mapped

to the STEP AP-214 format. The created DES specifications can be used for information exchange, simulation, verification as well as automatic controller synthesis.

1 Introduction

In the light of rapidly changing market needs, demands on flexibility and ability to decrease time to market, while still maintaining, or preferably increasing, product quality at a low cost, is becoming increasingly important. One step towards decreasing time to market is the use of a more efficient information exchange between product design and manufacturing systems design. Making information about product design solutions instantly available to engineers involved in manufacturing systems design would lead to a much shortened iteration cycle. More specifically, this would entail a process planner starting the documentation of how to manufacture a product already during the product design phase, based on preliminary design solutions. The produced process plan can then be used as a base for simulating how the introduction of a new product will affect an existing, or new, manufacturing system. The outcome of that simulation will also influence the final design solution of the product, as well as the manufacturing system. In more detail the process plan defines process information as a set of product-, process- and resource characteristics, defining what to produce, and how to produce it.

The present research, which is a more thorough and complete description of the ideas presented in (Falkman, Nielsen and Lennartson 2003b), aims at making use of an international standard, ISO10303-214, the application protocol 214 (AP214) of the STEP-standard (STandard for Exchange of Product model data) (TC184/SC4 2001), for communication and storing of process specifications. In the beginning STEP AP-214 was developed to represent product information. In recent years, however, it has been extended to include both process- and resource information as well. Even though there are a lot of software tools available for the generation of both product- and resource information, e.g. PDM systems, CAD systems, and robot simulation systems, there is still lack of tools for producing process information. Therefore, such a tool is presented in this paper. This tool makes use of a language based on process algebra and Petri nets, introduced in (Falkman and Lennartson 2001, Falkman and Lennartson 2005a), for the generation of process-specifications with special emphasis on sequence of operations and resource allocation. This high level language for discrete event systems (DESs), called process algebra Petri net (PPN), has been developed in accordance with the process relations defined in STEP-AP214. The aim of the present paper is to show how process specifications created with the PPN tool can be mapped to the extended STEP AP-214 format. The presented mapping defines the relationship between the information structure and the DES specification.

Much research has already been conducted both on information modelling, e.g. (Schenk and Wilson 1994), (Scheller 1990), and (Eversheim et al. 1991), and discrete event modelling, e.g. (Cassandras and Lafortune 1999), (Hoare 1985). However, little has been investigated concerning the connection between information and DES models, i.e. how an information structure could be mapped to a discrete event structure of a process plan. The created DES models will be used for automatic controller synthesis.

Controller synthesis includes two levels of control descriptions; the resource allocation system which takes care of the synchronization of common resources, and a detailed control of the individual resources. In the resource allocation system a number of products utilize a number of shared resources which are to be booked and unbooked. In this paper, the PPN language is utilized with the intention of simplifying the specification of desired routes. The detailed control, also specified using the PPN language, involves the specific control of each resource.

The approach is validated through a case study at Volvo Car Corporation, Torslanda, Sweden. This is done by creating PPN-specifications based on the Volvo data and then implementing the mapping method in order to automatically generate an ISO 10303-214 model. In the following two sections an introduction is given to both the PPN specification language and the ISO10303-214 standard. In addition to this, a comparison is made between the information and the discrete event models, with respect to the product, process, and manufacturing resource representations in both PPN and ISO10303-214 (AP214).

2 Discrete Event Modelling Language

DESs are systems that, at any given moment in time, is in one state out of a set of states and changes state according to the occurrence of one event out of a finite set of events. DESs can be modelled using a number of different modelling languages e.g. Petri nets (Peterson 1981), process algebra (Hoare 1985, Milner 1989), automata theory (Hopcroft and Ullman 1979, Kozen 1997) etc. This paper uses a specification language called process algebra Petri net (PPN), presented in detail in (Falkman and Lennartson 2005a), which combines Peri nets and process algebra. It is a further development of the MPPN language introduced in (Falkman and Lennartson 2001).

2.1 Process Algebra Petri Net

In the present paper only a short introduction to the PPN language is given. This language combines the graphical features of Petri nets with the compact expressions of process algebra in order to realize a powerful specification tool for DESs. The transition between two Petri net places in a PPN is a process P. A process $P = a_1 \rightarrow P_1$ describes that, first the event a_1 occurs, then it behaves like a process P_1. A number of process operators are now defined. These operators are used together with Petri nets in order to realize compact DES models.

Alternative The alternative operator $+$ specifies that there is a choice between two processes. Let two processes be defined as $P = a_1 \rightarrow P_1$ and $Q = b_1 \rightarrow Q_1$. Then an alternative between these two processes is described as

$$P + Q = a_1 \rightarrow P_1 \mid b_1 \rightarrow Q_1 \tag{1}$$

using Hoare's (Hoare 1985) choice symbol $|$. This implies that either event a_1 occurs followed by process P_1 or event b_1 occurs followed by process Q_1.

Arbitrary order Arbitrary order describes that all processes involved are to be executed, but the order does not matter. We define an arbitrary order operator $\oplus$ for n processes as

$$\oplus\{P_1, \ldots, P_n\} = \text{perm}(P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n) \tag{2}$$

with perm denoting the sum of all different deterministic permutations of the sequence $P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n$. For $n = 3$ this means that $perm(P_1....) = P_1 \rightarrow (P_2 \rightarrow P_3 + P_3 \rightarrow P_2) + P_2 \rightarrow$

$(P_1{\rightarrow}P_3+P_3{\rightarrow}P_1)+P_3{\rightarrow}(P_1{\rightarrow}P_2+P_2{\rightarrow}P_1)$. For arbitrary n a recursive function which generates a deterministic process expression is given in (Falkman and Lennartson 2005a).

Synchronization The nonstandard synchronization operator $\&$ implies that the first and last event of involved processes are to be synchronized, with no respect to common events. Again, consider two processes $P = a_1 \rightarrow P_1$ and $Q = b_1 \rightarrow Q_1$. The synchronization between P_1 and P_2 is denoted

$$P_1 \& P_2 \tag{3}$$

This means that a_1 in P occurs at the same time as b_1 in Q. This synchronized event is denoted $a_1 \& b_1$. If a process only involves a single event then this event will only be synchronized with the first events from the other processes. The synchronization operator $\&$ is especially useful when specifying resource allocation systems. For such systems it is desired to specify that an operation requires a certain resource, i.e. book the resource so that noone else can use it at the same time. Using the synchronization operator between an operation and a resource specifies that the booking of a resource is synchronized with the first event of the operation and the unbooking of the resource is synchronized with the last event of the operation. This is in Fig. 1 exemplified with an operation $O = c \rightarrow d \rightarrow e$ and a resource model $R = R_1{\rightarrow}R$ where $R_1 = b{\rightarrow}u$. The synchronized execution of O and R_1 is specified using the PPN langauge in Fig. 1a and with synchronized events in Fig. 1b.

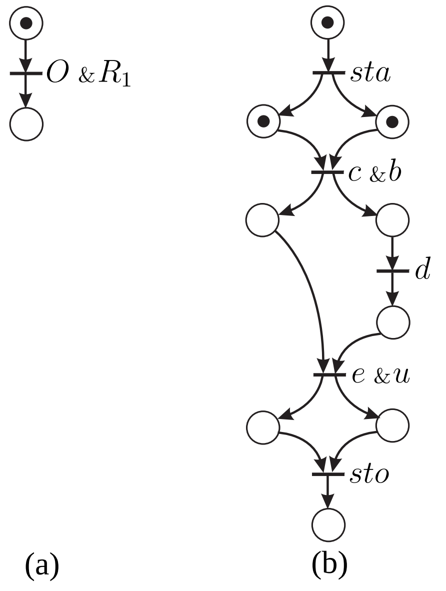


Figure 1: The process $P = O$ & R_1 given in (a) as a PPN model and in (b) with synchronized events.

Parallel processes A parallel execution offers the possibility for processes to execute independently of each other. The expression

$$P = \| \{P_1, \ldots, P_n\} \qquad (P_i \quad in \quad parallel) \tag{4}$$

denotes a process P which executes all processes P_i in parallel. If the alphabets of all involved processes are disjunct their events will be executed independently, resulting in interleaving, cf. (Hoare 1985). If, on the other hand, processes have common events these events have to be synchronized. The operator $\Vert$ is an extension of Hoares full synchronous composition (FSC) (Hoare 1985), involving the explicit process synchronization operator $\&$.

Start and stop operators Start and stop operators denote the starting and ending of the execution of processes. The start operator declares that a process will start by performing its first event. It is therefore used to specify when a process is allowed to start. In the same way the stop operator tells that a process will finish by performing its last event. Consequently it can be used to control when a process is permitted to finish.

In Fig. 2 the sequence $P_1^{\uparrow} \oplus P_2^{\downarrow}$ is illustrated. This expression says that either $P_1 = a \to b$ starts its execution before $P_2 = c \to d$ finishes by executing its last event or vice versa. In this case the continuation of process P_1 is not specified.

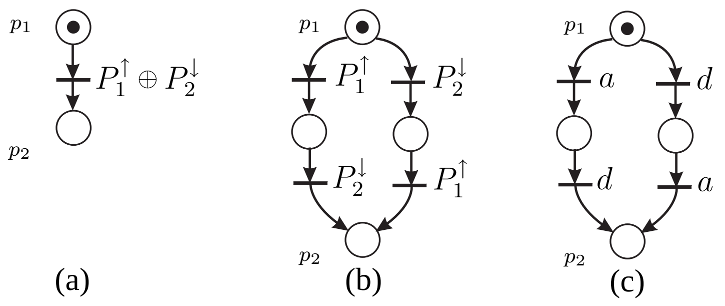


Figure 2: The process $P = P_1^{\uparrow} \oplus P_2^{\downarrow}$ given in (a) and (b) as PPN models and in (c) as a PN where $P_1 = a \to b$ and $P_2 = c \to d$.

Restrictions Restrictions when an operation can begin and/or end its execution are described by logical expressions introduced inside square brackets, [], following the process to be restricted. The logical expressions specify a specific state (e.g. an initial state) or a set of states. Common logic operators, $\wedge$ (and), $\vee$ (or), and $\overline{P}$ (inverse) are allowed inside these expressions.

In the present paper two basic atomic restrictions are used. The first is $[P]$, which describes that process P has to be in its initial state. The second is $[P^{\downarrow}]$, which specifies that P is in its final state. Also note that $[\overline{P}]$ refers to all states except the initial one, and $[\overline{P^{\downarrow}}]$ implies all states except the final one.

The expression

$$P_1[P_2^{\downarrow} \vee P_3^{\downarrow}] \tag{5}$$

specifies that a process P_1 is not allowed to start before process P_2 or process P_3 have finished their executions. In other words, P_2 or P_3 must be in its final state before P_1 can begin.

2.2 Process Operation Model

The focus of this paper is on specification of *resource allocation systems*. Such systems, within a manufacturing system, typically involve a set of products that share a set of resources. To ensure that only one product at a time is using a specific resource it is necessary for each resource to be *booked* by a specific product. It is also important to control that there are no *blocking* or *deadlock states* (Coffman, Elphick and Shoshani 1971). In this paper a so-called *high level routing specification* (HRS) will be used to specify a products route through a resource system. This HRS describes which operations a product is to undergo, in which order these operation are to be executed, and which resource(s) that may be used for each individual operation.

The PPN language is a general language for modelling DES, but will in the present paper be used for modelling high level routing specifications of resource allocation systems. A high level routing specification, modelled as a PPN, can also formally be converted into an automata representation, which is described in detail in (Falkman and Lennartson 2005b). The booking and unbooking specification, given as a number of automata, may then be used for verification and simulation as well as formal controller synthesis, e.g. (Ramadge and Wonham 1989), (Åkesson et al. 2003).

The high level routing specification involves two parts; relations of operations (ROP), see Example 16, and safety specifications. The ROP describes the possible operation sequences, taking into account necessary resources and predecessor demands. The predecessor demands define which operations that have to be executed before others can follow. Safety specifications are described on the same high level as the ROP, i.e. as sequences of operations. However, these safety specifications are created through a *specification synthesis*. The *specification synthesis* is performed on two different types of models; execution of operation (EOP) and interlocking specifications (IL), see Example 16. The execution of operation is a detailed description of each operation and the interlocking specifications involve restrictions on specific events in an EOP-specification. The *specification synthesis* converts the interlocking specifications and the EOP specifications into safety specifications describing operation sequences, necessary in order to guarantee the interlocking specifications. A more detailed description of the different parts included in the high level routing specification can be found in (Richardsson 2005), and detailed information about the specification synthesis is given in (Andersson et al. 2005).

Example 16 – High level routing specification using PPN. An ROP specification is in Fig. 3(a) given as a PPN. The ROP involves six operations and describes two parallel paths. The first operation, O_1, requires resource R_1. Operations O_2 and O_3 are to be executed in arbitrary order. These two operations require R_2 and R_3 respectively. Operations O_4 and O_5 are to be executed in parallel and require R_4 and R_5 respectively. Operation O_6 can use either R_6 or R_7, but cannot start its execution until operation O_3 has finished its execution. A resource consists of a number of sub-resources, also called components (Richardsson 2005). Resource R_1 in this example includes two components R_{1a} and R_{1b}. Both these components can be in states off and on. In Fig. 3(b) an EOP specification of operation O_1 is presented. This operation describes a sequence of three events $turnoff_{R_{1a}}$, $turnoff_{R_{1b}}$, and $finish$. The event $turnoff_{R_{1a}}$ executes when both R_{1a} and R_{1b} are in their on state. The next event $turnoff_{R_{1b}}$ executes when R_{1a} has changed state to off, while the operation is finished as soon as R_{1b} has changed state to off. Fig. 3(c) describes an interlocking for the event $turnoff_{R_{1a}}$, which specifies that another resource R_2 has to be in state on before $turnoff_{R_{1a}}$ can execute.

□

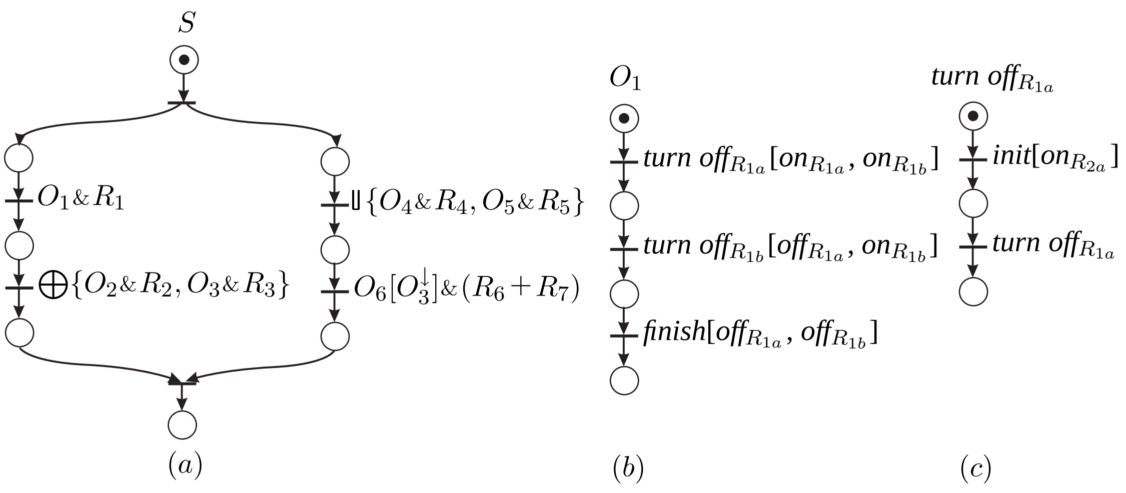

S
$O_1\&R_1$
$\bigoplus\{O_2\&R_2, O_3\&R_3\}$
$\Cup\{O_4\&R_4, O_5\&R_5\}$
$O_6[O_3^{\downarrow}]\&(R_6+R_7)$
O_1
turn off$_{R_{1a}}$[*on*$_{R_{1a}}$, *on*$_{R_{1b}}$]
turn off$_{R_{1b}}$[*off*$_{R_{1a}}$, *on*$_{R_{1b}}$]
finish[*off*$_{R_{1a}}$, *off*$_{R_{1b}}$]
turn off$_{R_{1a}}$
init[*on*$_{R_{2a}}$]
turn off$_{R_{1a}}$
(a)
(b)
(c)

Figure 3: (a) An ROP given as a PPN, (b) an EOP describing operation O_1, and (c) an interlocking for event *turn off*$_{R_{1a}}$.

2.3 Resource and Product Model

A model of a manufacturing system, considered as a resource allocation system, is created by synchronizing all of the involved resource models. The different resources may be modelled as two state models with two events, the booking and the unbooking events. In order to achieve a deterministic resource allocation model, however, a specific booked place for each product is included as in Example 17.

Example 17 – Resource and routing specifications A resource is modelled as a recursive process, see Fig. 4(c). Assume that $R_\ell^i = b_\ell^i \to u_\ell^i$, where b_ℓ^i represents the booking of resource R_ℓ by product i and event u_ℓ^i represents the corresponding unbooking event. A corresponding PN for R_ℓ where $R_\ell^1 = a_\ell^1 \to b_\ell^1$ and $R_\ell^2 = a_\ell^2 \to b_\ell^2$ is given in Fig. 4(d).

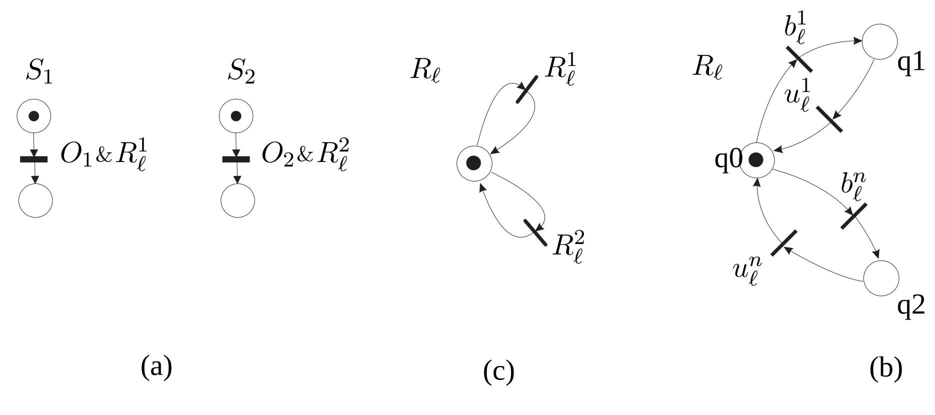


Figure 4: (a) Two routing specifications given as HRSs, (b) a resource model given as a PPN model and (c) the same resource with explicit booked places, $q0$ and $q1$, for each routing specification.

Specification S_1 in Fig. 4(a) is a HRS description that includes one operation O_1 and specifies that resource R_ℓ is required. The utilization of resource R_ℓ is therefore controlled by the resource model which models the mutual exclusion between S_1 and S_2. For instance, if the booking event b_ℓ^1 is synchronized with the first event of S_1, then the token in the resource model moves from place $q0$ to $q1$. S_2 is now unable to perform the synchronized process $O_2 \& R_\ell^2$ and book R_ℓ since b_ℓ^2 can not be executed because the token in the resource model needs to be in $q0$. Hence, S_2 has to wait to book R_ℓ until S_1 has unbooked R_ℓ.

□

The static information for a product, e.g. product id, may be assigned to the token in the PPN model. Note that this may involve colored Petri nets (Murata 1989), but this extension will not be emphasized in the present paper.

3 Standard for Information Exchange

The objective of ISO 10303 standard also called STEP is to provide the framework for the unambiguous representation and exchange of computer interpretable product data throughout the lifecycle of a product (Herzog and Torne 2001). The STEP standard is divided into a number of application protocols. Application protocols standardize the use of STEP to support a particular manufacturing function reliably and consistently (Trapp 1993).

3.1 The STEP AP214 Model

In this section the product, process, and manufacturing resource (PPR) representation in ISO 10303-214 (AP214) will be described. However, this will not be a complete description, rather the objective is to provide a description of where information needed to generate the PPN-specification will/can be found.

ISO 10303 STEP is an international standard that "provides a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged" (TC184/SC4 1994). The term *exchange* should be interpreted as the exchange of data between computer systems in environments associated with the complete life-cycle of a product, including manufacturing. As an introduction to the STEP standard the following concepts will be explained:

- application protocol(AP),
- application reference model (ARM),
- application interpreted model (AIM),
- integrated resource (IR), and
- the exchange/sharing mechanisms.

AP: Application protocols, define the scope, context, and information requirements for a particular application, e.g. the automotive industry (AP214), or the electrical design and installation (AP212). An application protocol is divided into two different representations of the information requirements: the application reference model (ARM) and the application interpreted model (AIM). In depth information on application protocols and STEP in general are available in (Warthen 1990, Kemmerer 1999, Owen 1993).

ARM & AIM: The ARM is used to capture the information requirements using application-based terminology i.e. terminology that is understood by the domain experts of that particular application. For instance, in AP214 terms like *wheel space* and *overall axle distance* would be used, because they are widely used in the automotive industry.

AIM uses neutral resource constructs specified by the IRs to define the same requirements, i.e. the neutral resource constructs are *interpreted* to meet the requirements defined in the ARM. In addition, the AIM provides a mechanism for inter-operability between different application protocols to, e.g. describe mechatronic products by using both AP214 and AP212. In this paper the ARM model of AP214 is used.

AP214 is an application protocol developed to consider the automotive industry requirements on information exchange. However, (Johansson 2001b) has shown that the generic structure of AP214 can be used to represent any type of mechanical product, including a manufacturing resource.

EXPRESS The EXPRESS language (Schenk and Wilson 1994) is a formally specified and structured language used to define the ARM models in STEP. The EXPRESS language is an earlier and more general alternative to the Unified Modelling Language (UML). Usually, the ARM is also defined in EXPRESS and often presented in EXPRESS-G, a graphical subset of the EXPRESS Language.

The basic constructs of EXPRESS (and EXPRESS-G) is the entity and the attribute. An entity is similar to a class in object-oriented programming, i.e. it is a representation of something of interest in the real world. The attribute is a kind of property, and as such it represents a particular aspect of an entity.

Graphically, in EXPRESS-G, an entity is represented as a box with a name in it, cf. Fig. 5. The name is the identifier of the item it represents in the real world. Attributes are rep-

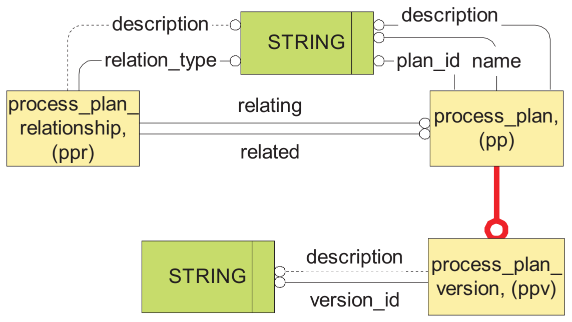


Figure 5: Example of entities represented in the EXPRESS language.

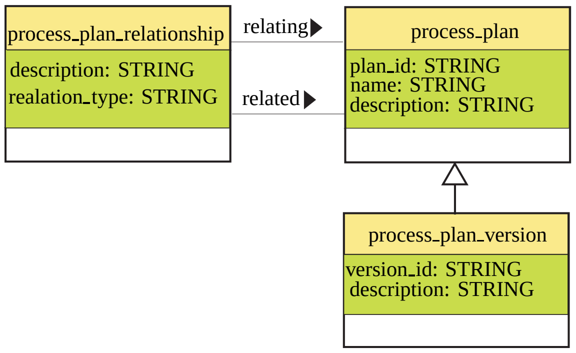


Figure 6: The same model as in 5 given as a UML model.

resented by a line ending with a small circle, showing the direction of the relationship. They are labelled with the name of the attribute as well as any cardinality constraints. A dashed line represents an optional attribute whereas a thick line represents a supertype-subtype relationship, i.e. the same as inheritance in object-oriented programming e.g. Unified Modelling

Language (UML) (Booch et al. 1999), cf. Fig. 6. A supertype, i.e. the parent of an inheritance relationship, can be abstract (ABS) meaning that the entity can not be populated with data.

The model in Fig. 5 is a conceptual model and will be used to present the AP214 standard. Another type of model will also be used to exemplify the use of AP214, the instantiated model, which is a model populated with data from the real world, cf. example in Fig. 7. The triangle in the lower right corner of the entities in Fig. 7 indicates that it is an instantiated model. In some instances a filled triangle will occur, indicating that all mandatory attributes are instantiated, for example the *process_plan_relationship* entity in Fig. 7. A transparent triangle on the other hand indicates that some, or all, of the mandatory attributes have been left out.

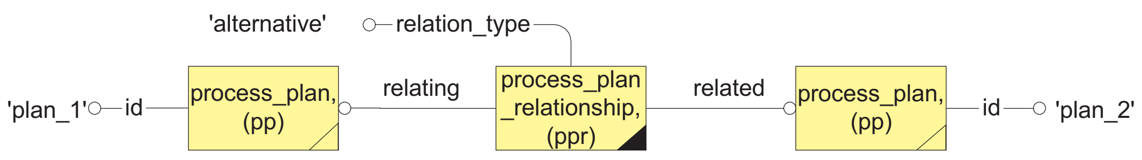


Figure 7: Instantiated models of the model in Fig. 5. Two process_plans relating to each other by an alternative relationship which means that ”plan_2” is an alternative process_plan to ”plan_1”.

Parts included in AP214 Fig. 8 shows the different parts of the design and production project that STEP AP214 includes (Johansson 2001a). As can be seen in this figure three different main areas are described: product, process, and resource (manufacturing system). *Control code* and *Behavior models* are not included in the standard and are therefore not marked in gray.

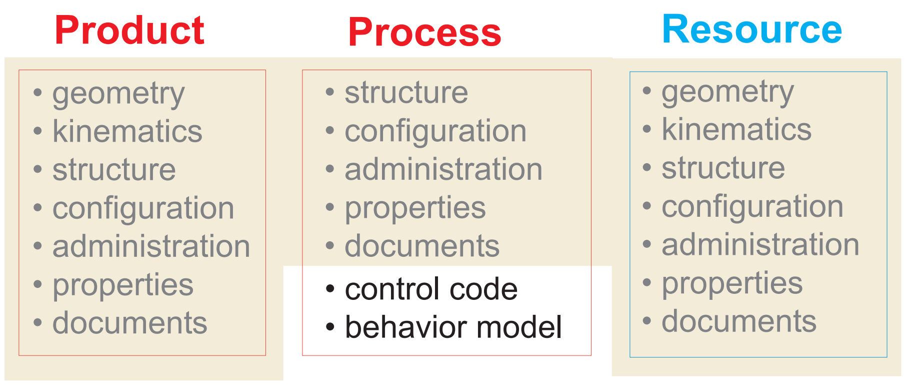


Figure 8: Graph describing three different parts in STEP AP214. These parts are product, process and resource. All the information marked in gray are included in the standard. Not included is therefore *control code* and *behavior models* in the process part.

In order to create the *behavior* model and the *control code* which is not described in STEP, using the information given by STEP standard, a mapping i necessary. The mapping maps information from a discrete event model into the product, process and resource part of the STEP standard in Fig. 8. The PPN-specification, which models the DES, can then be used in order to create *control code* and *behavior models*.

3.2 Process Operation Model

The process model in AP214, cf. Fig. 9, plays a central role when representing a PPN-specification. It is the holder of all the necessary process information, such as the process plan identifier, relationships between processes, sequences etc. The process model consists of a structure to hold meta-data about a process plan. This structure is identified by the *process_plan* (pp) in Fig. 9.

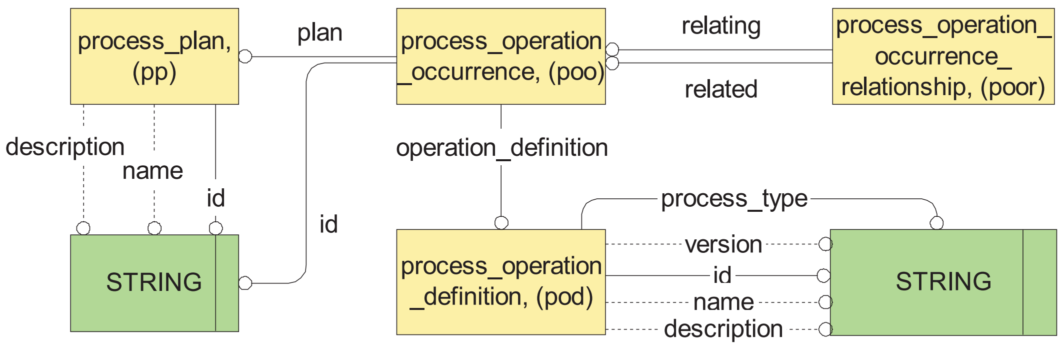


Figure 9: Representation of process data in AP214. A *process_operation_occurence* (poo) has a *process_operation_definition* (pod), a *process_plan* (pp), and a *process_operation_occurrence_relationship* (poor).

A *process_plan* consists of one or more processes represented by the *process_operation_occurrence* (poo). The *process_operation_occurrence* represents the occurrence of a process in a process plan. More specifically, it represents the occurrence of a definition of a process, the *process_operation_definition* (pod). This mechanism enables the reuse of a definition in several different places in a process plan, as well as in several different process plans. This makes it possible for different versions of the same plan to reuse definitions that have not been changed from the former version. For instance, alternative resource for the same operation would be represented by the same definition, but with different resources assigned to different *process_operation_occurrence*s, all representing the same definition.

The relationship between two *process_operation_occurences* is represented by the *process_operation_occurrence_relationship* (poor) where the attribute *relation_type* holds the type of relationship. The attribute *relating* points in the direction of the *process_operation_occurrence* prior to the *process_operation_occurrence* pointed out by the attribute *related*. Thus Fig. 10 shows two processes with the *relation_type* equal to ”sequence”. This implies that process with *id*=”O1” executes first followed by a second process with *id*=”O2”.

The process information needed in order to represent a PPN-specification is *process_plan.id* cf. Fig. 9, *process_ operation_occurrence.id*, *process_operation_definition.name*, and

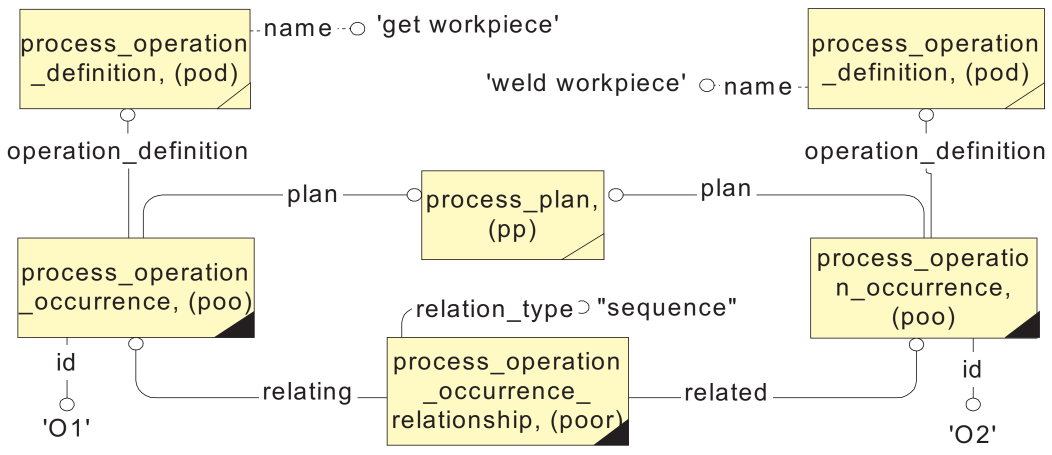

process_operation _definition, (pod)
name
'get workpiece'
'weld workpiece'
name
process_operation _definition, (pod)
operation_definition
operation_definition
plan
process_plan, (pp)
plan
process_operation _occurrence, (poo)
relation_type
"sequence"
process_operatio n_occurrence, (poo)
id
'O1'
relating
process_operation _occurrence_ relationship, (poor)
related
id
'O2'

Figure 10: Populated process model representing the sequence between two processes.

the information about the relationships between processes.

3.3 Resource and Product Model

Resource Model The manufacturing resources can be represented in several different ways in AP214, depending on the level of detail and the design life cycle stage. Two different representations have been identified as important with respect to a PPN-specification, cf. Fig. 11, the *single_instance* (si) and the *physical_instance* (pi).

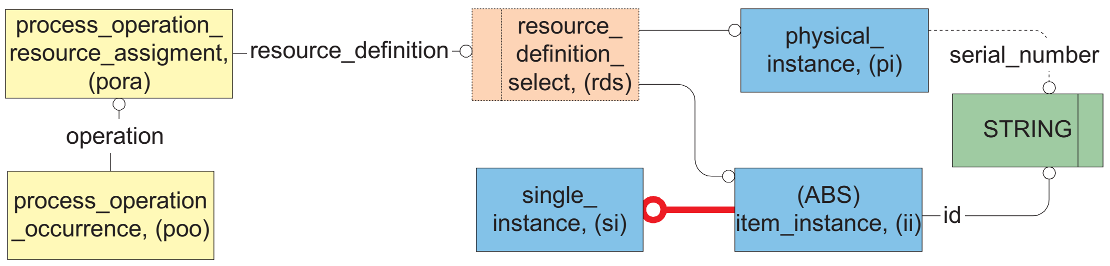

process_operation_ resource_assigment, (pora)
resource_definition
resource_ definition_ select, (rds)
physical_ instance, (pi)
serial_number
STRING
operation
process_operation _occurrence, (poo)
single_ instance, (si)
(ABS) item_instance, (ii)
id

Figure 11: Representation of resource data in AP214.

The *single_instance* and the *physical_instance* are both instances of an abstract representation of a manufacturing resource (*item*), but there is a significant difference between them. The *single_instance* represents one occurrence of a certain type of manufacturing resource, whereas the *physical_instance* represents a physical resource on the shop floor. Thus the *single_instance* is better suited for planning purposes before a physical resource exists, while the *physical_instance* is better used when there already exists a physical resource. A select type is used to model that there is a selection between the two different types of instances. The resource_definition relation refer to the resource_definition_select which in turn refers to the two different instances. The term (ABS) is used to indicate that item_instance is an abstract supertype to single_instance. This means that it cannot exist in itself, only by virtue of its subtypes. However, the abstract supertype is useful since it enables attributes to be collected at a higher level within the data model and then inherited.

The attributes of the manufacturing resources that are needed in order to generate a PPN-specification are the *single_instance.id* and the *physical_ instance.serial_number*, cf. Fig. 11.

Product Model The most important product entities in AP214 are the *item* (i) and *item_ version* (iv). These are the holders of product meta-data, such as identifiers, version data, classification data and much more, cf Fig. 12.

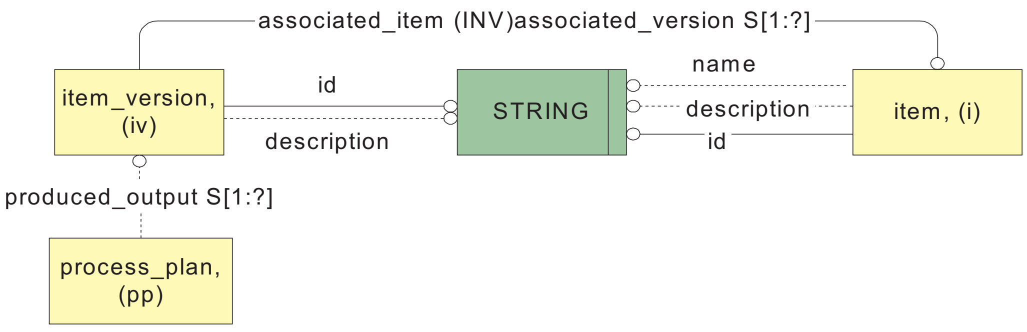


Figure 12: Entities where meta-data about a product and a resource (item) is represented in AP214. In this figure two new relation types are used. Enumeration relation S[1:?] describes that a *process_plan* may refer to one or many *item_versions*. Inverse relation (INV) describes that an *item* may refer to one or many *item_versions* which is the inverse direction of the relation as it is written.

For the purpose of creating a product in the PPN-specification, however, only the product identifier is needed. The product identifier is represented by the *item.id*. The product identifier is related to process information via the *process_plan.produced_output* as shown in Fig. 12. As can be seen an enumeration relation S[1:?] is used in the relation *produced_output*, which describes that a *process_plan* may have one or many *item_versions*. An *item_version* may only have one *item*. However, an *item* may refer to one or many *item_versions* which is modelled by the inverse relation (INV) together with the enumeration relation S[1:?]. The inverse relation is used in order to obtain a complete relation between two entities, meaning that the relation consists of both the normal direction and also the inverse direction.

3.4 Operation and Interlocking Model

The detailed information required when creating EOP and interlocking specifications, cf. Fig. 3, can be found in Fig. 13. The main attribute is *condition_assignment*, which connects the *process_operation_occurrence* with resource information, ie. *physical_instance*, via *state_assignment*. The *condition_assignment* defines a state condition that has to be fulfilled before a *process_operation_occurrence* can be executed. A *condition_assignment* can refer to a number of *state_assignments*, describing that there could be more than one resource state that has to be fulfilled. The *condition_assignment* has an attribute ”name”, which can be either ”interlocking” or ”required resource state”. The ”required resource state” describes that the condition is for an EOP specification while naturally ”interlock” specifies that it belongs to an interlocking specification.

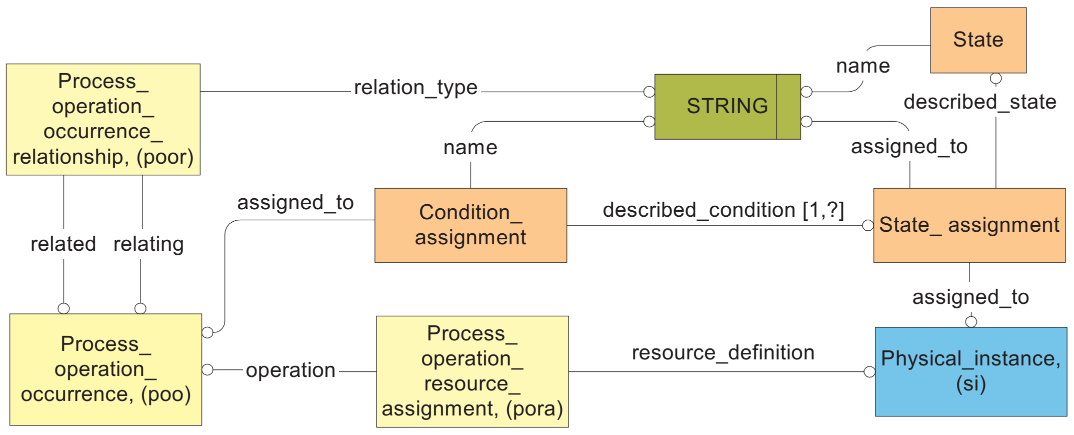


Figure 13: Representation of conditions for *process_operation_ocurrence* described in STEP.

The entity *state_assignment* refers to *state*, which holds the actual state information. *State_assignment* also refers to *physical_instance* so that the state can be associated with a specific resource. Note that because application protocol AP214 is used for representing processes, process relations, and resources, but is unable to represent resource states, it is necessary to also use AP239. In particular this includes the entities *State*, *State_assignment*, and *Condition_assignment* (von Euler-Chelpin et al. 2004). It is thus necessary to combine two application protocols in order to represent EOP and interlocking (IL) specifications.

4 Generation of Discrete Event Models for Control Synthesis and Verification

One of the main advantages of the suggested method of mapping information to and from an information exchange standard is that creation of product specifications for controller synthesis, verification and simulation can be performed with every manufacturer and industry, no matter what development systems they use as long as the information can be transferred according to the STEP standard. However, one should keep in mind that this mapping only involves part of all necessary information needed to control a cell or a plant. A large part of the necessary control code for the robot cell in the example presented in Section 5 also involves security information, fault handling etc. At this point however the focus is on flow control information, such as resource data, and involved operations together with the operation sequence. A natural extension of this method is therefore to also include security issues as well as fault handling etc. Table V.1 shows abbreviations for important entity names in STEP AP214 used in the mapping.

Entity names	**abbreviation**
process_plan	pp
process_plan_version	ppv
process_operation_occurence	poo
process_operation_definition	pod
process_operation_occurence_relationship	poor
process_operation_resource_assigment	pora
condition_assignment	ca
state_assignment	sa

Table V.1: Abbreviation table for important entity names in STEP AP214.

Mapping of AP214 into PPN In this section a definition of the relationship between AP214 and the PPN product, process and manufacturing system models is given. Examples are given in order to illustrate the mapping of the static description of the product to be manufactured into a DES model using the graphical notation of both descriptions. The different constructs are described separately. Table V.2 describes the relationship between the *relation_types* defined in STEP and the operators defined in the PPN language.

Relation_type in STEP	**Operators in PPN**
Sequence	Sequence $\rightarrow$
Substitution	Alternative $+$
Exclusiveness	Arbitrary order $\oplus$
Simultaneity	Parallel $\parallel$

Table V.2: Relationship between the *relation_types* defined in STEP and the operators defined in the PPN language.

A larger example in Section 5 is also given, describing the mapping of a welding process at Volvo Car Corporation, Torslanda, Sweden.

4.1 Process Operations

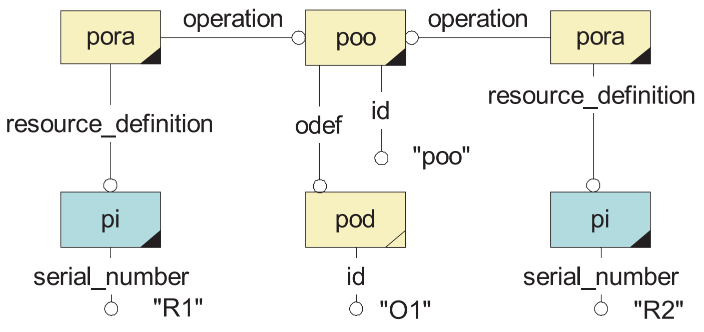
The necessary process information required in order to create a PPN-specification is the *pp.id*, *poo.id*, *pod.id*, and the information about relationships between processes. Table V.3 describes the use of AP214 process model information in order to create a PPN-specification. The differences between *pod* and *poo* are several, but the most important one in this paper is that *pod* gives a general description of what an operation involves, whereas the *poo* describes

AP214 Information	$\rightarrow$	**PPN-specification**
pp identifier	$\rightarrow$	PPN identifier
pod identifier	$\rightarrow$	operation identifier
poor type	$\rightarrow$	Net structure and process algebra operators

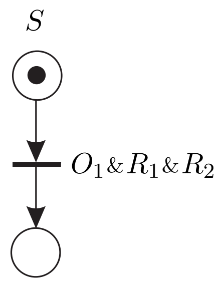
Table V.3: The use of AP214 information in creating a PPN-specification.

on a more detailed level which resources to use in a specific operation. For instance, a *pod* may be used to define two different *poos* in that they utilize different resources. Another difference is that several *poos* may be decomposed into more detailed *poos* and all those *poos* relate to the same *pod*. In the PPN a process that only differs in which resource it requires is regarded as the same operation. This means that it is natural to use the *product_operation_definition* identification in the translation into an HRS transition in PPN.

Multiple resource processes A multiple resource process describes that a process involves more than one resource which is naturally very common. In the STEP standard multiple resources is modelled as more than one *pora* referring to one *poo*. Each *pora* refers either to *pi* or an *si* which was described in Section 3.3. In Fig. 14(a) an example of multiple resources is shown. There are two *pora* that refers to a single *poo*. This implies that this



(a)



(b)

Figure 14: HRS describing multiple resources, in (a) as a STEP/Express model and in (b) as an a PPN.

poo requires two resources in order to be able to execute the specific process. Each *pora* refer to a single resource entity, *pi.serial_number* R_1 and *pi.serial_number* R_2 respectively. In the PPN-specification this is represented by a single operation O_1 with the two multiple resources R_1 and R_2 separated by the $\&$ operator, cf. Fig. 14(b).

Alternative resource processes Alternative resources, in order to perform the same process operation, are described by two or more *poos* that refer to the same *pod* and also refer to each other with the *poor.relation_type* 'substitution', cf Fig. 15(a). This means that the *pod* describes the main process to be performed and there are more than one *poo* that are able to perform this process. In PPN alternative resources are represented by a plus sign between the possible alternative resources in the transition equation, cf 15(b).

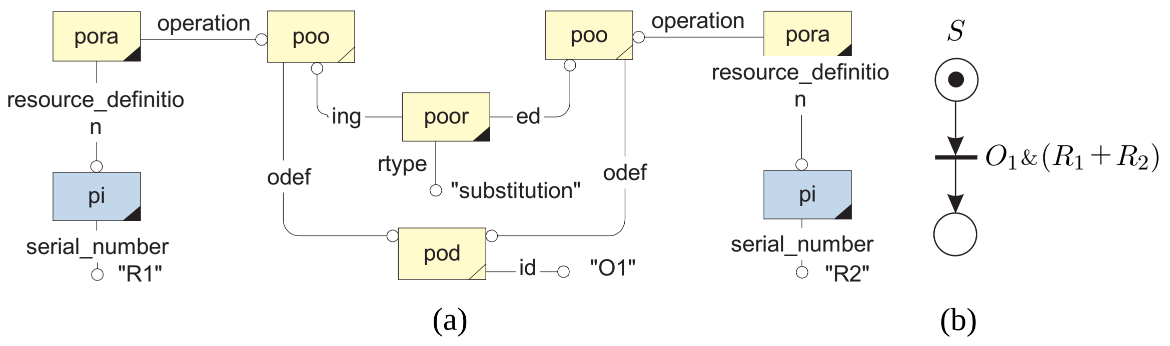


Figure 15: HRS describing alternative resources, in (a) as a STEP/Express model and in (b) as an a PPN.

Alternative processes (Substitution) Alternative processes are in STEP modelled as two *poos* that refers to each other with the *poor.relation_type* equal to 'Substitution'. The two *poos* do not refer to the same *pod*. This is described in more detail in (Falkman et al. 2003b).

Arbitrary order (Exclusiveness) Exclusiveness, or arbitrary order as it is denoted in PPN, is in STEP described by two or more *poos* that refers to each other with the attribute *relation_type* of *poor* equal to 'exclusiveness'. In Fig. 16 the same arbitrary ordered processes are represented in PPN using the $\oplus$ operator. The exclusiveness in STEP describes that there are more than one process that are to be performed and these processes may be conducted in arbitrary order. This in turn means practically that the order in which the processes are executed does not influence the product in the sense of quality, appearance etc.

Parallel (Simultaneity) Parallel processes can in the PPN language be described by either using Petri net semantics or by using the algebra operator $\Vert$, cf. Fig. 3. In STEP this is modelled as two or more *poos* referring to each other with the *poor.relation_type* 'simultaneity'. Parallel processes are conducted at the same time and can be exemplified by two parts that are refined separately in parallel with no concern to each other. After both have finished there individual refinement they are joined in some way to finish the product with identity ppv.

Decomposition Decomposition of an operation specify a detailed sequence of events. Each of these events specify a change in state for a resource. Detailed operation descriptions are in the PPN language given as (EOP specifications), cf. Fig. 3. A decomposed operation in STEP involves information about what the specific resources are meant to do during the process, e.g. open, move to position, weld a spot. The decomposition relation is specified by a *poo* referring to another *poo* by the *poor* with *relation_type* equal to 'decomposition', cf Fig.

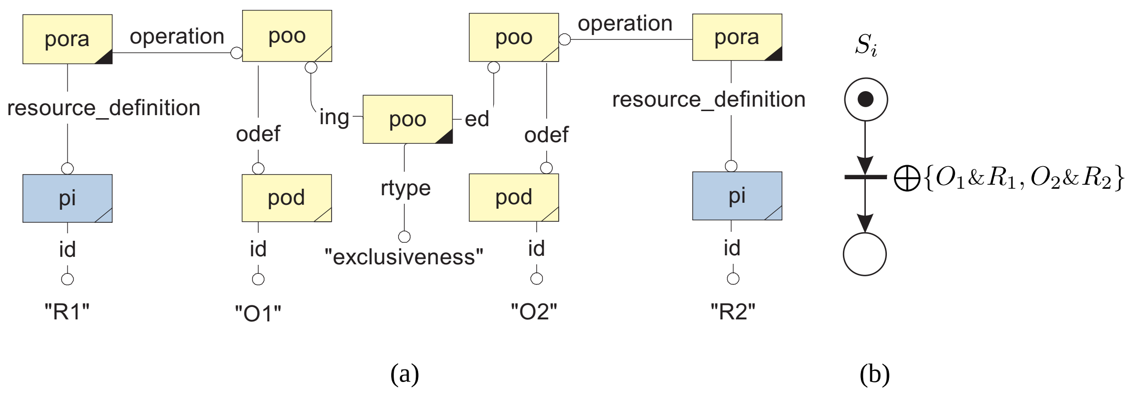


Figure 16: HRS describing arbitrary ordered processes, in (a) as a STEP/Express model and in (b) as an a PPN.

17(a). The example given in Section 16 is in Fig. 17 using the STEP standard. This is a very simplified decomposed control sequence just to demonstrate the principle. In Section 4.3 a complete EOP specification with resource states is presented.

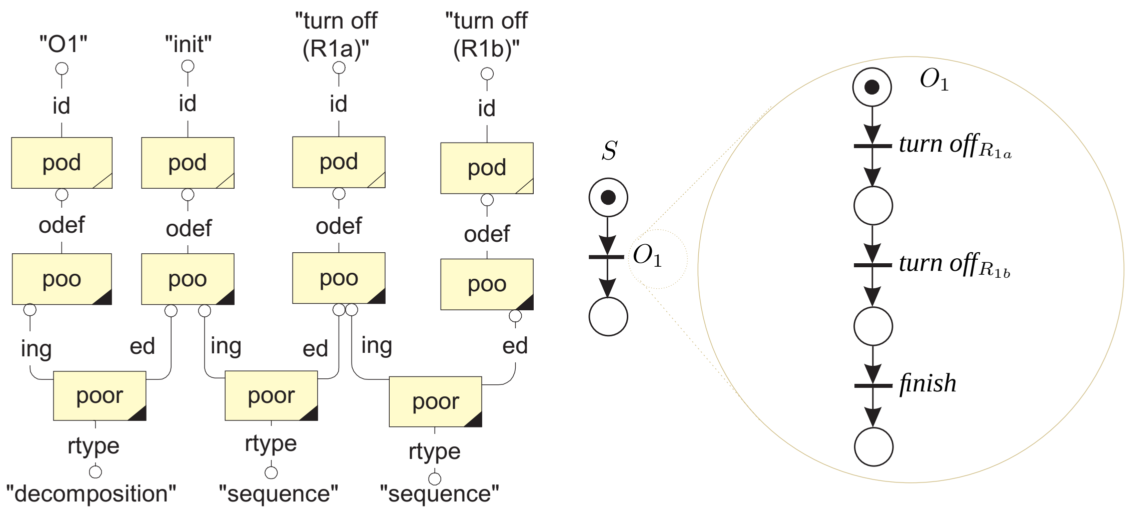


Figure 17: In (a) a STEP/Express model is given describing the decomposition of operation O_1, no resources are included. In (b) a corresponding PPN is shown, specifying the decomposition of operation O_1.

4.2 Resources and Products

The attribute of a manufacturing resource that is needed in order to generate a PPN-specification is either the *si.id* or the *pi.serial_number*, cf. Fig. 11 and Fig. 15.

For the purpose of creating a product in the PPN-specification only the product identifier is needed. The product identifier is represented by the *iv.id* in AP214. The product identifier is related to process information via the *pp.produced_output* as shown in Fig. 12.

4.3 Operation and interlocking

Operation description As described in Example 16, resource states are used in order to specify the execution of an EOP specification. In STEP this means that a *condition_assignment* refers to a *process_operation_occurrence* with the attribute *name* equal to ”required resource state”, see Fig. 13. The *condition_assignment* refers to one or more *state_assignment* via *described_condition*. Resource states are described in STEP by *state* with the attribute *name*. The *state_assignment* refers to both *state* and *physical_instance*, which connects a resource state to a specific resource. In Fig. 18 one transition of the EOP specification in Fig. 3 is presented. A *poo* refers to a *pod* with *id* equal to ”*turn off*”. Two *condition_assignments* with *name* equal to ”required resource state” are referring to this *poo*. The two required states are described by the entities *state* and the related resources are described by *physical_instance*.

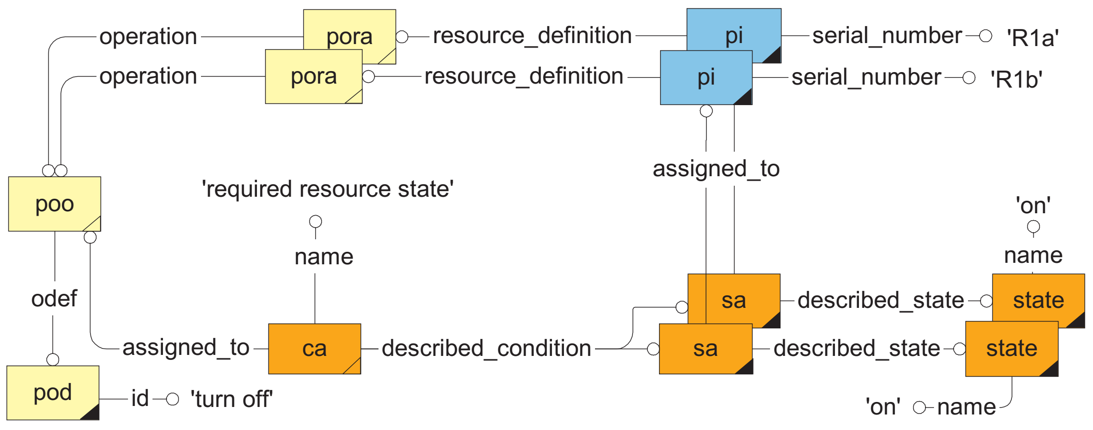


Figure 18: Operation described by populated STEP model.

Interlocking description A special kind of condition is the interlocking, described in Example 16. This condition restricts when an event in the EOP specification is allowed to execute, cf. Fig. 3. The interlocking model in STEP is very similar to the operation description just described in the previous section. The difference is that the attribute *name* for *condition_assignment* is ”interlock”. There can be multiple as well as alternative interlock conditions, which is shown in the example in Section 5. In Fig. 19 the interlocking specification in Fig.3(c) is presented as a STEP model. A *condition_assignment* with *name* equal to ”interlock” refers to a *poo*, which refers to a *pod* with *id* equal to ”*turn off*”. The *state* describes the specific state ”on” and *physical_instance* specifies the related resource, in this case ”R_2”.

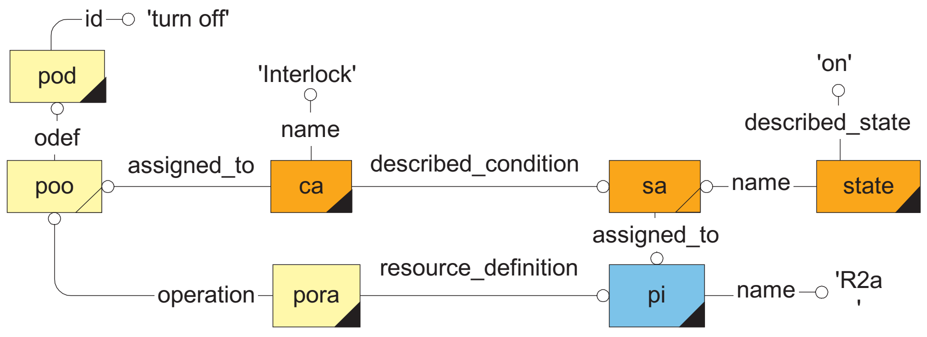


Figure 19: Interlocking described by populated STEP model.

5 Example: A robot cell in the Volvo factory

The following example describes how a resource allocation system may easily be modelled by use of the PPN language. It is also demonstrated how the PPN specifications may be represented in the STEP standard. The PPN specification consists of three different parts, as described in Example 16. These are; a *relation of operation* (ROP) specification, *execution of operation* (EOP) specifications, and *interlocking* (IL) specifications. The resource allocation system is a robot cell at Volvo Car Corporation, Torslanda, Sweden and is illustrated in Figure 20. This robot cell consists of four robots R_1-R_4, two Fixtures F_1-F_2, two turntables T_1-T_2, and a conveyor C_1.

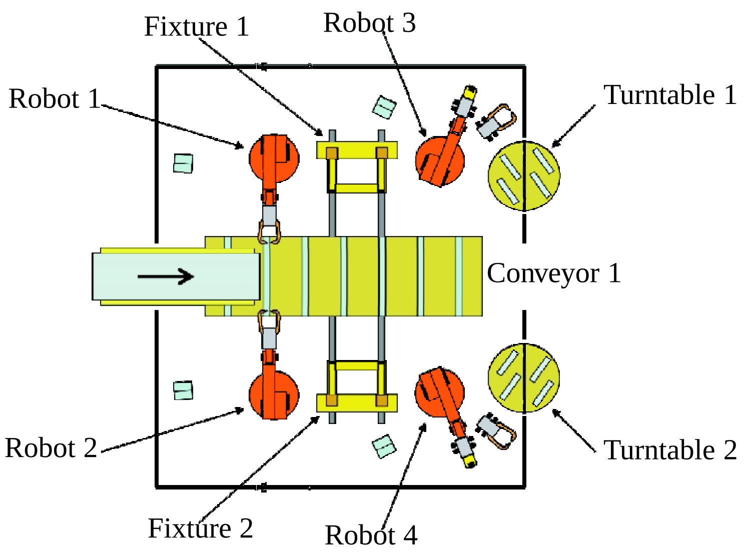


Figure 20: A welding cell at Volvo Car Corporation, Torslanda, Sweden.

The task of the example cell in Figure 20 is to weld a plate to the side of the floor of the car, underneath the doors. This operation is to be executed on both car models produced in the cell. The models are, Volvo V70, and S80. During a work cycle Robot 4 picks a part from the rack on Turntable 2 and then places it in Fixture 2. Simultaneously Robot 2 starts to weld previously loaded, but not completely welded, parts of the body. Robot 4 changes tool from gripper to weld gun, Fixture 2 positions the plate on the body, and both Robot 2 and Robot 4 weld the new part to the body.

Relations of operations specification First we have to create the ROP specification, describing the general sequence of operations. This is done in a way that guarantees as much flexibility as possible in the cell, but at the same time fulfilling given restrictions such as predecessors. When creating a specification using the PPN language a choice has to be made regarding the use of Petri net constructs versus algebra expressions. For some systems it might be advantageous to use Petri net constructs to a large extent, while other systems might benefit more from making greater use of algebra expressions. In order to realize an easy-to-read and concise specification, this decision is up to each individual specifier.

In Fig. 21 part of the ROP for Volvo V70 is presented. It involves eight operations divided into three parallel paths using the Petri net construct for parallel execution. The first operation in each parallel path does not have any predecessors and it is therefore reasonable to assume that these can execute in parallel. The parallel execution of operations O_2 and O_3 is, however, modelled using the algebra expression. This is done since both operation O_2 and operation O_3 are required to be completed before operation O_4 is allowed to be executed. Operation O_4 is followed by the execution of either operation O_{5A} or O_{5B}, which in its turn is followed by the execution of operation O_7. There is a restriction on operation O_7 specifying that this operation is not allowed to execute before operation O_6 has finished its execution. Operation O_6 is followed by operation O_{17}. The reservation of necessary resources for each operation is specified by synchronizing the execution of every operation with corresponding resource model, cf Example. 17.

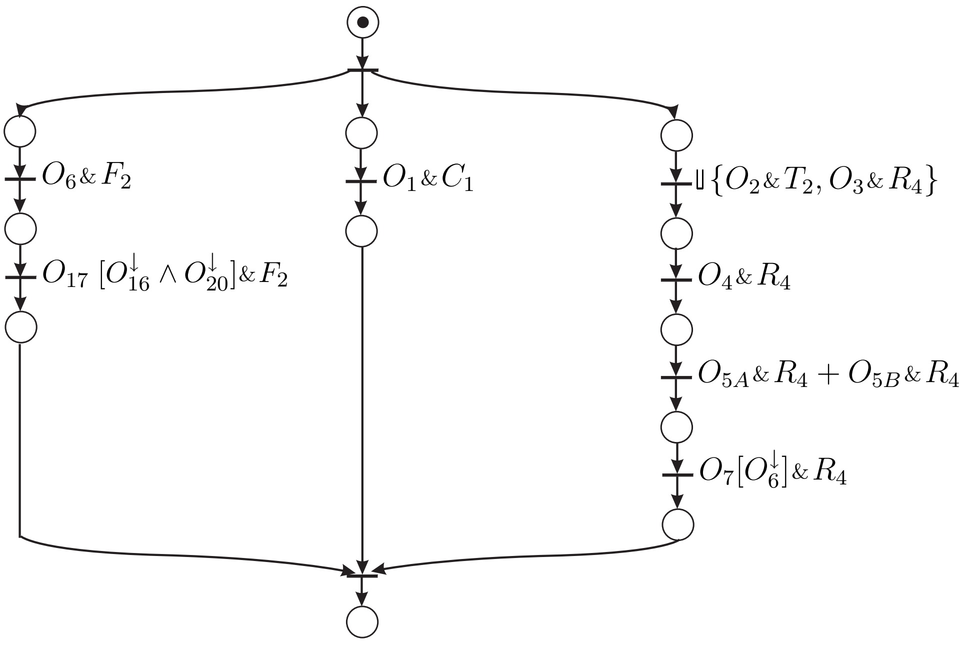

$O_6 \& F_2$
$O_{17}\ [O_{16}^{\downarrow} \wedge O_{20}^{\downarrow}] \& F_2$
$O_1 \& C_1$
$\Vert\{O_2 \& T_2, O_3 \& R_4\}$
$O_4 \& R_4$
$O_{5A} \& R_4 + O_{5B} \& R_4$
$O_7[O_6^{\downarrow}] \& R_4$

Figure 21: Specification of the relation of operations (ROP) given as an PPN model.

The ROP specification in Fig. 21 is modelled as a STEP/Express model in Fig. 22. Note that the resources in the PPN model are not included in the STEP/Express model. The

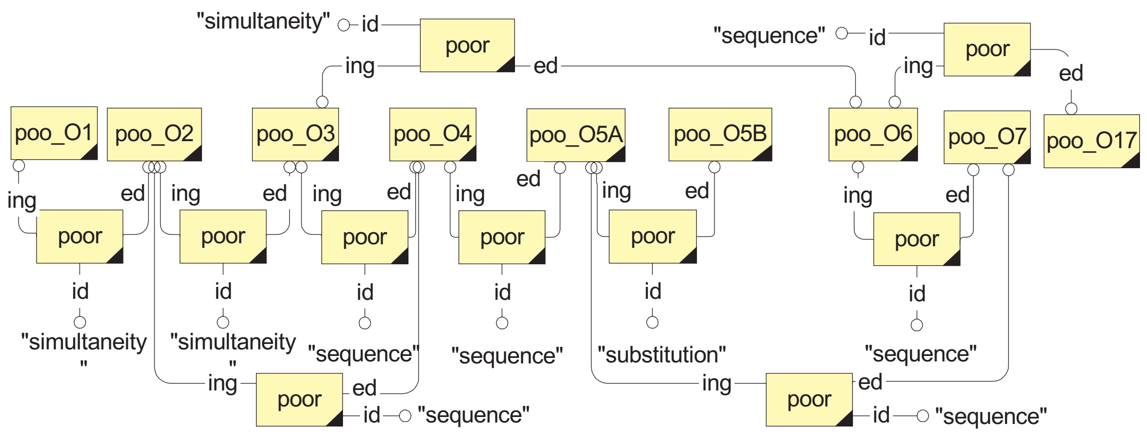


Figure 22: The ROP specification in Fig. 21 given as an instantiated STEP/Express model, no resources included in the figure nor are operations O_{16} and O_{20}.

parallel execution of operations O_1, O_2, O_3, and O_6 is specified using the *relation_type* equal to ”simultaneity”. The fact that operation O_4 has to wait until both operations O_2 and O_3 have finished is specified in Fig. 22 using the *relation_type* equal to ”sequence” between both O_2 and O_4 as well as between O_3 and O_4. Operation O_7 follows both operation O_6 and either of O_{5A} or O_{5B}.

Execution of operation specification Each individual operation is given as a PPN specification. The operations are described by changes of the involved resource states.

The EOP specification for operation O_{17} is presented in Fig. 24. This operation moves the fixture to its home position. The EOP specification for this operation specifies a sequence of three events that the operation executes. The resource fixture F_2 involves five different components, Y_{18}, Y_{16}, Y_{14}, $SG3$, and $SG6$, whos states constitute state conditions for each event. This means that in order for each event to occur, in the EOP specification in Fig. 23, these components have to be in certain states. The first event *open*$_{Y_{14}}$ executes when Y_{18} is in state *work pos*, Y_{16} is in state *locked*, Y_{14} is in state *closed*, $SG3$ is in state *on*, and $SG6$ is in state *on*. Component Y_{14} is the clamp on the fixture and event *open*$_{Y_{14}}$ specifies the opening of the clamp, which in turn means that Y_{14} is to change state to *open*, cf. Fig. 23. The second event $unlock_{Y_{16}}$ is performed when the clamp has changed state. The finishing of the operation is specified by the last event $finish$, which occurs when Y_{16} has changed state to $unlocked$.

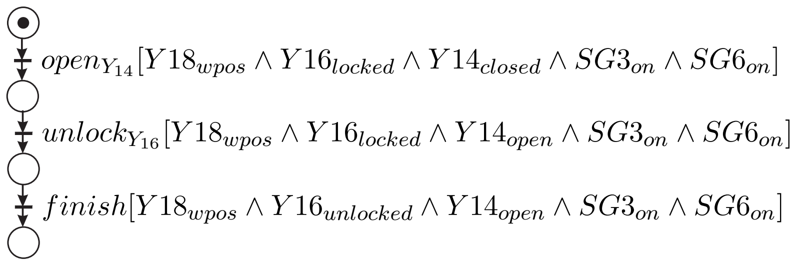

$open_{Y_{14}}[Y18_{wpos} \wedge Y16_{locked} \wedge Y14_{closed} \wedge SG3_{on} \wedge SG6_{on}]$
$unlock_{Y_{16}}[Y18_{wpos} \wedge Y16_{locked} \wedge Y14_{open} \wedge SG3_{on} \wedge SG6_{on}]$
$finish[Y18_{wpos} \wedge Y16_{unlocked} \wedge Y14_{open} \wedge SG3_{on} \wedge SG6_{on}]$

Figure 23: EOP specification of operation OP_{17} given as a PPN.

The EOP specification described in Fig. 23 as a PPN is in Fig. 24 described as a STEP/ Express model. The five components are described by the *physical_instances* with the attribute *id* equal to the components identity. Every *pi* refers to a *pora*, which in turn refers to a *poo*, as described in Section 3.3. A *condition_assignment* refers to the state *state_assignments*, which refers both to the specific resource as well as its state.

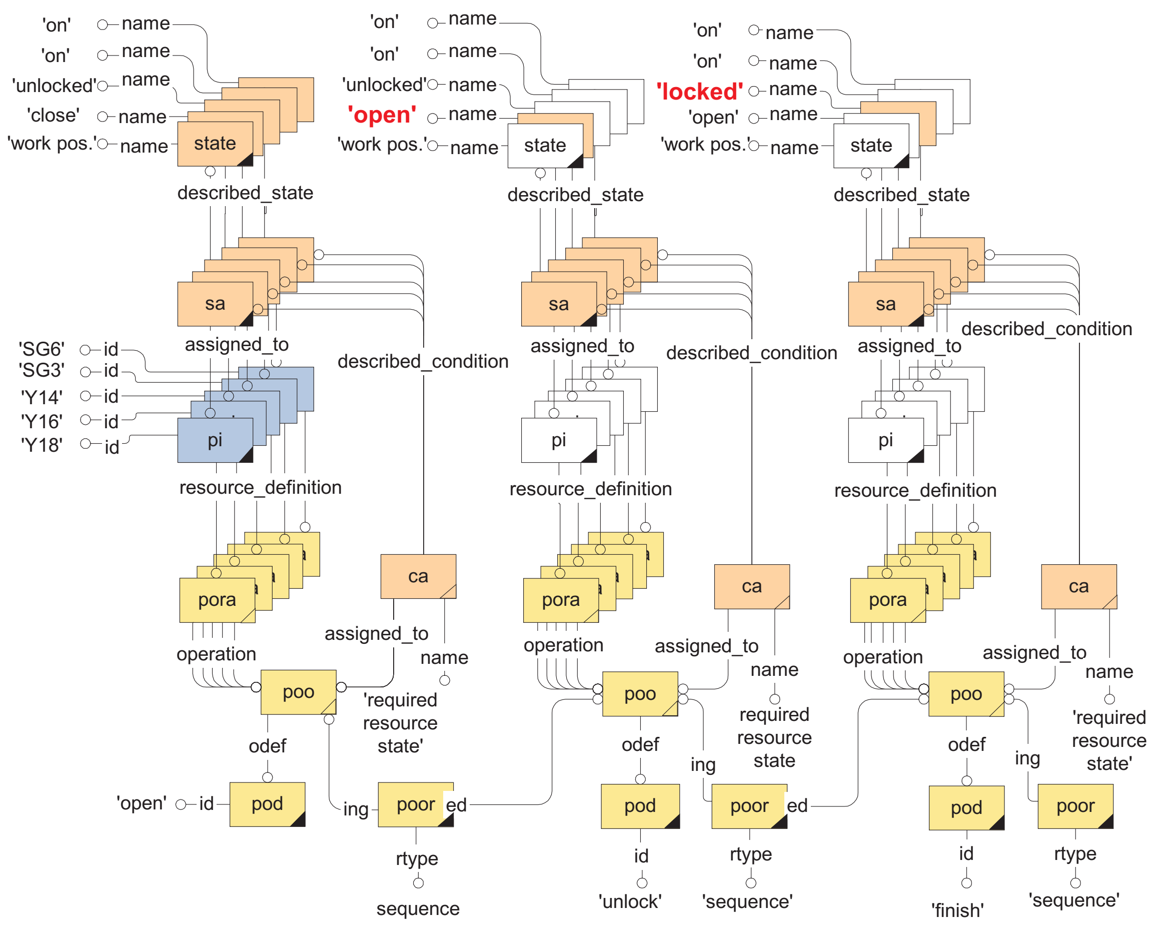


Figure 24: EOP specification of operation OP_{17} given as an instantiated STEP/Express model.

Interlocking specification As can be seen in Fig.23 the only component that changes state between the initial state and the first action is the clamp Y_{14}. Fig. 25 shows the interlock when Y_{14} goes from closed to open, which is modelled with the event $open_{Y_{14}}$ in the EOP specification, cf. Fig 23. Fig. 25 describes two alternative requirements, and one of these has to be fulfilled before Y_{14} is allowed to change state to *open*. The first requirement specifies that Y_{18} is in its home state. The second alternative requirement specifies that if Y_{18} is in its work position state then O_{16} cannot be ongoing, i.e. O_{16} is in its initial or final state. This is in Fig. 25 modelled as a restriction on the event *init*, where all the necessary state conditions have to be fulfilled before it is possible for $open_{Y_{14}}$ to be executed.

The interlock specification in Fig. 25 is specified using a STEP/Express model in Fig. 26. In this figure three alternative restrictions for event $open_{Y_{14}}$ are described using three *condi-*

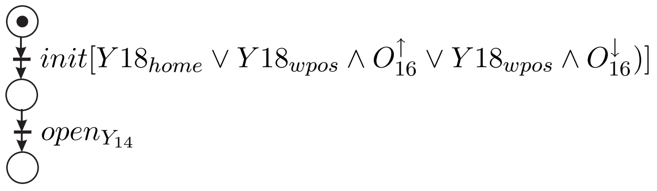

$init[Y18_{home} \vee Y18_{wpos} \wedge O_{16}^{\uparrow} \vee Y18_{wpos} \wedge O_{16}^{\downarrow})]$
$open_{Y_{14}}$

Figure 25: Interlock specification for *open*$_{Y_{14}}$ given as a PPN.

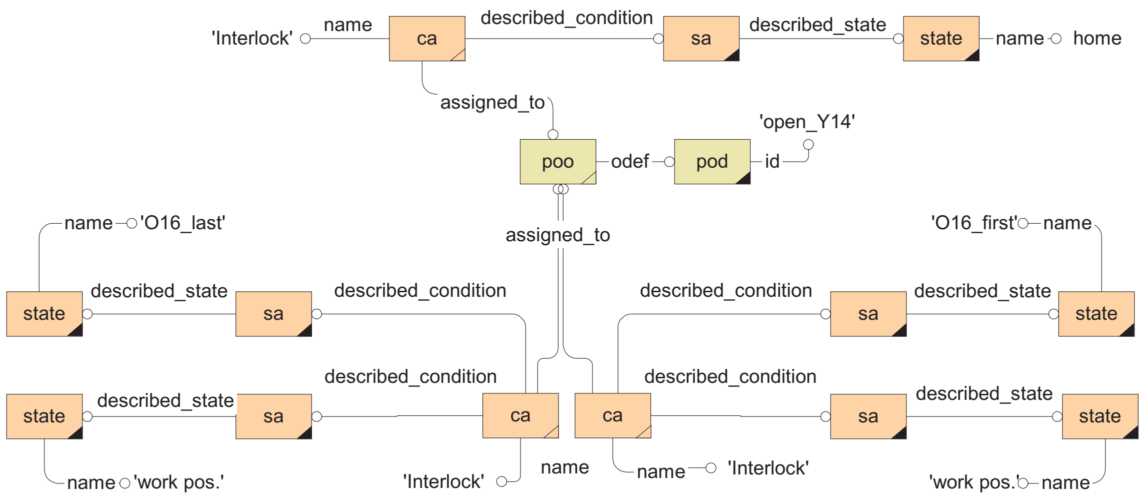


Figure 26: Interlock specification for *open*$_{Y_{14}}$ described as a STEP/Express model.

tion_assignment entities. The attribute *name* is in each of these *condition_assignments* equal to ”interlock” and they all refer to the same *poo*. The required states specified by each *condition_assignment* are described by the *state* via the *state_assignment*. Note that the resource entities *pora* and *pi* are left out of Fig. 26 in order to increase clarity.

6 Conclusions

This paper has shown how the well accepted international standard STEP-AP214 can be used for communication and storing of resource allocation system specifications. A tool for creating such specifications has also been briefly presented, based on a mapping between the STEP information model and the resource allocation system specification. This mapping implies a reliable framework for the exchange of control related information involving resource, product and process information.

The presented tool uses a language called process algebra Petri nets (PPN), introduced in (Falkman and Lennartson 2005a), for generating process-specifications, e.g. the specifications of resource allocation systems. The PPN language has been developed in accordance with the process relations defined in STEP-AP214. It has been shown how this tool can be used to specify complex systems in a compact, but yet highly readable manner.

A resource allocation system involves three parts, high level routing specifications, resource models, and a supervisor. The high level routing specification involves three specifications (Richardsson 2005), relations of operations (ROP), execution of operation (EOP), and interlocks (IL). The present research, which is a more thorough and complete description of the ideas that were presented in (Falkman et al. 2003b), has illustrated how the PPN language as well as the STEP standard can be used for high level specification but also for detailed operation specifications, such as execution of operations (EOP) and interlocking (IL) specifications (Richardsson 2005).

The method has been partly implemented and the result has been validated using case studies at both Scania Oskarshamn, Sweden (Falkman et al. 2003b) and Volvo Torslanda, Sweden. In future work the entire method will be implemented and applied to large industry cases.

References

Åkesson, K., Fabian, M., Flordal, H. and Vahidi, A. (2003). Supremica - a tool for verification and synthesis of discrete event supervisors, *11th Mediterranean Conference on Control and Automation*, Rhodos, Greece.

Andersson, K., Richarsson, J., Lennartsson, B. and Fabian, M. (2005). Hierarchical control applying information reuse and supervisor synthesis, *To be submitted to Transactions on Automation Science and Engineering* .

Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The Unified Modeling Language User Guide*, Addison-Wesley object technology series, , ISSN 99-2816663-3, Addison-Wesley, Harlow.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, Kluwer Academic Publishers.

Coffman, E., Elphick, M. and Shoshani, A. (1971). System deadlocks, *Computing Surveys* **3**(2): 67–78.

Eversheim, W., Marczinski, G. and Cremer, R. (1991). Structured modelling of manufacturing processes as nc-data preparation, In Annals of the CIRP, volume 40/1.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P. and Lennartson, B. (2005a). A high level specification language based on process algebra and petri nets, *To be submitted to Transactions on Automation Science and Engineering* .

Falkman, P. and Lennartson, B. (2005b). Using a high level language for verification and control synthesis of discrete event systems, *Submitted to Transaction on Control System Technology* .

Falkman, P., Nielsen, J. and Lennartson, B. (2003). Automatic generation of object models for process planning and control purposes using an international standard for information exchange, *Journal on Systemics, Cybernetics and Informatics* **1**(5).

Herzog, E. and Torne, A. (2001). Information modelling for system specification representation and data exchange, *Proc. of 8th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems.*, Washington, DC, USA, pp. 136–143.

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Series in Computer Science, Addison-Wesley.

Johansson, M. (2001a). Personal comunication at presentation of doctorial thesis, - http://dictionary.reference.com/search?r=2&q=dissertation, Royal Institute of Technology, Department of Production Engineering.

Johansson, M. (2001b). *Information Management for Manufacturing System Development*, PhD thesis, Kungliga Tekniska Högskolan. ISSN: 1650-1888.

Kemmerer, S. (1999). *Step - the grand experience*, NIST special publication, 939. edited by Sharon J. Kemmerer.

Kozen, D. (1997). *Automata and Computability*, ISBN 0-387-94907-0, Springer-verlag New York, inc.

Milner, R. (1989). *Communication and Concurrency*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Murata, T. (1989). Petri nets: properties, analysis and applications, *Proc IEEE* **77**(4): 541–580.

Owen, J. (1993). *STEP - A introduction.*, ISBN 1-874728-04-6, Information Geometers, 1st edition, Winchester.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems, *Proc. IEEE* **77**(1): 81–98.

Richardsson, J. (2005). *Development and Verification of Control Systems for Flexible Automation*, Licentiate thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 015.

Scheller, A. (1990). Information modeling for distributed applications, *Proc of second IEEE Workshop on Future Trends of Distributed Computing Systems*.

Schenk, D. and Wilson, P. (1994). *Information Modeling: The EXPRESS Way*, ISBN 0-19-508714-3, Oxford University Press.

TC184/SC4, I. (1994). Iso 10303-1: Industrial automation systems and integration - product data representation - and exchange - part 1: Overview and fundamental principles, ISO Standard.

TC184/SC4, I. (2001). Iso 10303-214: Industrial automation systems and integration - product data representation - and exchange - part 214: Core data for automotive mechanical design processes, ISO Standard.

Trapp, G. (1993). *The emerging STEP standard for product-model data exchange*, Volume: 26, issue 2, Computer.

von Euler-Chelpin, A., Holmstrm, P. and Richardsson, J. (2004). A neutral representation of process and resource information of an assembly cell supporting control code development, process planning and resource life cycle management, *2nd International Seminar on Digital Enterprise Technology*, Seattle, USA.

Warthen, B. (1990). Application protocols - step access, *Product Data Int'l,Warthen Comm.*, **1**(2): 5–7.

references

bibliography

Aceta, L., Larsen, K. and Ingólfsdóttir, A. (2004). An introduction to milners ccs.

Adlemo, A., Andreasson, S., Fabian, M., Gullander, P., Hellgren, A., Lennartson, B., Liljenvall, T. and Pernebo, L. (1997). Models for specification and control of flexible manufacturing systems, *Technical report*, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report nr CTH/RT/R-97/003.

Åkesson, K. (2002). *Methods and tools in supervisory control theory*, Phd thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 431.

Åkesson, K., Fabian, M., Flordal, H. and Vahidi, A. (2003). Supremica - a tool for verification and synthesis of discrete event supervisors, *11th Mediterranean Conference on Control and Automation*, Rhodos, Greece.

Åkesson, K., Flordal, H. and Fabian, M. (2002a). Exploiting modularity for synthesis and verification of supervisors, *Proc. of 15'th IFAC World Congress on Automatic Control*, Barcelona, Spain.

Åkesson, K., Flordal, H. and Fabian, M. (2002b). Exploiting modularity for synthesis and verification of supervisors, *Proc. of the IFAC World Congress on Automatic Control.*, Barcelona, Spain.

Alenljung, T. and Lennartson, B. (2005). Simplified modeling of manufacturing systems - an introduction to sensor activation graphs, *Proc. of IEEE International Conference on Automation Science and Engineering*, Edmonton, Canada, pp. 261– 266.

Andersson, K., Richarsson, J., Lennartsson, B. and Fabian, M. (2005). Hierarchical control applying information reuse and supervisor synthesis, *To be submitted to Transactions on Automation Science and Engineering* .

Arnold, A. (1994). *Finite Transition Systems: Semantics of Communicating Systems*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Basten, T. (1998). *In Terms of Nets:System Design with Petri Nets and Process Algebra*, PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication, *Information and Control* **60(1-3)**: 109–137.

Bergstra, J. A. and Klop, J. W. (1985). Algebra of communicating processes with abstraction., *Theor. Comput. Sci.* **37**: 77–121.

Bergstra, J. and Klop, J. (1982). Strong normalization and perpetual reductions in the lambda calculus, *Elektronische Informationsverabeitung und Kybernetik* **18**: 403417.

Best, E., Devillers, R. and Koutny, M. (1998). Petri nets, process algebras and concurent programming languages, *Proc of ICM'98*, Berlin, Germany.

Best, E., Devillers, R. and Koutny, M. (2001). *Petri net algebra*, EATCS monographs on theoretical computer science, Springer, Berlin.

Best, E., Devillers, R. and Koutny, M. (2002). The box algebra = petri nets + process expressions, *Information and Computation* (178): 44–100.

Bloom, B., Cheng, A. and Dsouza, A. (1997). Using a protean language to enchance expressiveness in specification, *IEEE Transactions on Software Engineering* **23**(4): 224–234.

Bolognesi, T. and Brinksma, E. (1987). Introduction to the iso specification language lotos, *Computer Networks and ISDN Systems* **14**(1): 25–59.

Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The Unified Modeling Language User Guide*, Addison-Wesley object technology series, , ISSN 99-2816663-3, Addison-Wesley, Harlow.

Brinksma, E. (1995). Performance and formal design: a process algebraic perspective, *Proc. of Sixth International Workshop on Petri Nets and Performance Models*, IEEE, Durham, NC USA, pp. 124 – 125.

Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, Kluwer Academic Publishers.

Coffman, E., Elphick, M. and Shoshani, A. (1971). System deadlocks, *Computing Surveys* **3**(2): 67–78.

Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2001). *Introduction to Algorithms, Second Edition*, 2nd edn, The MIT Press.

Crow, J., Vito, B. D., Lutz, R., Roberts, L., Feather, M. and Kelly, J. (2005). *Formal methods specification and analysis guidebook for the verification of software and computer systems.*, URL http://eis.jpl.nasa.gov/quality/Formal_Methods/. Volume II: A practitioners companion.

David, R. and Alla, H. (1992). *Petri Nets and Grafcet*, Prentice Hall International (UK) Ltd, Hertfordshire HP2 4RG.

Degano, P., DeNicola, R. and Montanari, U. (1987). Ccs is an (augmented) contact-free c/e system, *in* E. M. Venturini Zilli (ed.), *Mathematical Models for the semantics of Parallelism*, Vol. Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 144–165.

Dictionary of Algorithms and Data Structures, process algebra (2004).
URL: *http://www.nist.gov/dads/HTML/processalgbr.html*

Eversheim, W., Marczinski, G. and Cremer, R. (1991). Structured modelling of manufacturing processes as nc-data preparation, In Annals of the CIRP, volume 40/1.

Fabian, M. and Lennartson, B. (1994). Petri nets and control synthesis; an object oriented approach., *Proc of the 2nd IFAC/IFIP/IFORS Workshop on Intelligent Manufacturing Systems, IMS '94*, Vienna, Austria.

Falkman, P. and Lennartson, B. (2001). Combined process algebra and petri nets for specification of resource booking problems, *2001 IEEE American Control Conference*, Arlington, VA, USA.

Falkman, P. and Lennartson, B. (2005a). A high level specification language based on process algebra and petri nets, *To be submitted to Transactions on Automation Science and Engineering* .

Falkman, P. and Lennartson, B. (2005b). Using a high level language for verification and control synthesis of discrete event systems, *Submitted to Transaction on Control System Technology* .

Falkman, P., Lennartson, B. and Tittus, M. (2001). Modeling and specification of discrete event systems using combined process algebra, *Proc. of 2001 IEEE/ASME Advanced Intelligent Mecatronics*, COMO, Italy.

Falkman, P., Lennartson, B. and Tittus, M. (2005). Specification of a batch plant using process algebra and petri nets, *To be submitted to Transactions on Control Engineering Practice* .

Falkman, P., Nielsen, J. and Lennartson, B. (2002). A formal mapping of static information models into dynamic models for process planning and control purposes, *Proc. of WODES 2002*, Spain.

Falkman, P., Nielsen, J. and Lennartson, B. (2003a). Automatic generation of object models for process planning and control purposes using an international standard for information exchange, *Proc. of SCI 2003*, Orlando, Florida, USA.

Falkman, P., Nielsen, J. and Lennartson, B. (2003b). Automatic generation of object models for process planning and control purposes using an international standard for information exchange, *Journal on Systemics, Cybernetics and Informatics* **1**(5).

Falkman, P., Nielsen, J. and Lennartson, B. (2004). A method for automated generation of discrete event systems from step ap214 for process planning and control, *Submitted to Journal of Manufacturing Systems* .

Harel, D., Pnueli, A., Schmidt, J. and Sherman, R. (1987). On the formal semantics of statecharts., *Proc. of Symposium on Logic in Computer Science.*, pp. 55–64.

Hellgren, A. (2000). *Modelling and Implementation Aspects of Supervisory Control*, Licentiate thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 350.

Hermanns, H., Herzog, U., Mertsiotakis, V. and Rettelbach, M. (1997). Exploiting stochastic process algebra achievements for generalized stochastic petri nets, *Proc. of International Workshop on Petri Nets and Performence Models*, IEEE, Los Alamitos, CA, USA, Saint Malo, Fr, pp. 183–192.

Herzog, E. and Torne, A. (2001). Information modelling for system specification representation and data exchange, *Proc. of 8th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems.*, Washington, DC, USA, pp. 136–143.

Heymann, M. and Meyer, G. (1991). An algebra of discrete event processes, *Technical report*, Ames Research Center, National Aeronautics and Space Administration.

Hoare, C. (1985). *Communicating Sequential Processes*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Hopcroft, J., Motwani, R. and Ullman, J. (2001). *Introduction to Automata Theory, Languages and Computation*, 2nd ed. edn, Addison-Wesley Series in Computer Science, Addison-Wesley.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Series in Computer Science, Addison-Wesley.

ISO 10303-1: Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles (1994). ISO standard.

Jmaiel, M. (2000). A unified algebraic framework for specifying communication protocols, *Proc. of Third international Conference on Formal Engineering Methods*, York UK, pp. 57–65.

Johansson, M. (2001a). Personal comunication at presentation of doctorial thesis, - http://dictionary.reference.com/search?r=2&q=dissertation, Royal Institute of Technology, Department of Production Engineering.

Johansson, M. (2001b). *Information Management for Manufacturing System Development*, PhD thesis, Kungliga Tekniska Högskolan. ISSN: 1650-1888.

Kemmerer, S. (1999). *Step - the grand experience*, NIST special publication, 939. edited by Sharon J. Kemmerer.

Kozen, D. (1997). *Automata and Computability*, ISBN 0-387-94907-0, Springer-verlag New York, inc.

Lennartson, B., Fabian, M., Tittus, M. and Hellgren, A. (1998). Modeling primitives for supervisory control, *Proc of WODES ’98*, Cagliari, Italy.

Mayr, R. (1997). Combining petri nets and pa-processes, *Theoretical Aspects of Computer Software (TACS’97), volume 1281 of Lecture Notes in Computer Science*, Sendai, Japan.

Milner, R. (1980). *A Calculus of Communicating Systems*, Vol. Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg New York.

Milner, R. (1989). *Communication and Concurrency*, International Series in Computer Science, Prentice–Hall International, Englewood Cliffs, NJ.

Minsky, M. L. (1989). *Computation: Finite and Infinite Machines*, Prentice-Hall, Hemel Hempstead.

Murata, T. (1989). Petri nets: properties, analysis and applications, *Proc IEEE* **77**(4): 541–580.

Olderog, E.-R. (1987). Petri nets and algebraic calculi of processes, *Advances in Petri Nets, 266 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany* pp. 196–223.

Olderog, E.-R. (1991). *Nets, Terms and Formulas*, Cambridge University Press, Trumpington Street, Cambridge CB2 1RP, Great Britain.

Owen, J. (1993). *STEP - A introduction.*, ISBN 1-874728-04-6, Information Geometers, 1st edition, Winchester.

Pena, M. and Cortadella, J. (1996). Combining process algebras and petri nets for the specification and systethis of asynchronious circuits, *Proc of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Fucushima, Japan.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc.

Plotkin, G. D. (1991). *A Structural Approach to Operational Semantics*, Computer Schience Department Aarhus University, DAIMI FN-19, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes, *SIAM J. Control Optim.* **25**(1): 206–230.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems, *Proc. IEEE* **77**(1): 81–98.

Reisig, W. (1985). *Petri Nets, An Introduction*, Springer Verlag.

Rescher, N. and Urquhart, A. (1971). Temporal logic, *Springer-Verlag, New York* .

Richardsson, J. (2005). *Development and Verification of Control Systems for Flexible Automation*, Licentiate thesis, Control and Automation Laboratory, Chalmers University of Technology, Göteborg, Sweden. Technical report 015.

Rondogiannis, P. and Cheng, M. (1994). Petri-net-based analysis of process algebra programs, *Elsevier Science Publisher B.P., Science of Computer Programming* pp. 55–89.

Scheller, A. (1990). Information modeling for distributed applications, *Proc of second IEEE Workshop on Future Trends of Distributed Computing Systems*.

Schenck, D. and Wilson, P. (1994). *Information Modeling: The EXPRESS Way*, Oxford University Press. ISBN: 0-19-508714-3.

Schenk, D. and Wilson, P. (1994). *Information Modeling: The EXPRESS Way*, ISBN 0-19-508714-3, Oxford University Press.

Stirling, C. (1996). *Logics for Concurrency: Structure versus automata*, Springer Verlag, chapter Modal and temporal logics for processes., pp. pp 149–237.

TC184/SC4, I. (1994). Iso 10303-1: Industrial automation systems and integration - product data representation - and exchange - part 1: Overview and fundamental principles, ISO Standard.

TC184/SC4, I. (2001). Iso 10303-214: Industrial automation systems and integration - product data representation - and exchange - part 214: Core data for automotive mechanical design processes, ISO Standard.

Trapp, G. (1993). *The emerging STEP standard for product-model data exchange*, Volume: 26, issue 2, Computer.

Vahidi, A., Lennartson, B. and Fabian, M. (2005). Efficient supervisory synthesis of large systems, *Control Engineering Practice*. Accepted.

von Euler-Chelpin, A., Holmstrm, P. and Richardsson, J. (2004). A neutral representation of process and resource information of an assembly cell supporting control code development, process planning and resource life cycle management, *2nd International Seminar on Digital Enterprise Technology*, Seattle, USA.

Warthen, B. (1990). Application protocols - step access, *Product Data Int'l,Warthen Comm.*, **1**(2): 5–7.