Eindhoven University of Technology

Department of Mathematics and Computing Science

Multi-dimensional Petri nets

by

W.M.P. van der Aalst

93/26

Computing Science Note 93/26
Eindhoven, August 1993

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

Multi-dimensional Petri nets

W.M.P. van der Aalst

Eindhoven University of Technology
Dept. of Mathematics and Computing Science

Abstract.
Petri nets have become a mature modelling and analysis tool applicable in many application domains. Since the introduction of the Petri net concept in 1962 (Petri [14]), Petri nets have been investigated and applied by many researchers. Many extensions of the classical Petri net have been proposed. To model attributes, 'colour' has been added. Time concepts have been added to describe the temporal behaviour of a system. In object-oriented Petri nets, tokens have a life-cycle and an identification.

The *multi-dimensional Petri net* model presented in this paper deals with these extensions in a unifying way. In this model an arbitrary number of dimension may be identified, e.g. a 'spatial dimension', a 'time dimension', a 'colour dimension', etc.

This paper also discusses the relations between the multi-dimensional Petri net model and other Petri net based models.

Multi-dimensional Petri nets can be analysed using traditional techniques. Moreover, it is possible to *project* a multi-dimensional Petri net onto a limited number of dimensions. By analysing the projected multi-dimensional Petri net we can deduce properties of the original multi-dimensional Petri net. Therefore, multi-dimensional Petri nets are also interesting from an analysis point of view.

1 Introduction

Petri nets have become a popular tool for modelling and analysis of concurrent systems. There are several factors which contribute to their success: the graphical nature, the ability to model parallel and distributed processes in a natural manner, the simplicity of the model and the firm mathematical foundation. Nevertheless, the classical Petri net model is not suitable for the modelling of many systems encountered in logistics, production, communication, flexible manufacturing and information processing. Petri nets describing real systems tend to be complex and extremely large. Sometimes it is even impossible to model the behaviour of the system accurately. To solve these problems, many extensions of the classical Petri net model have been proposed.

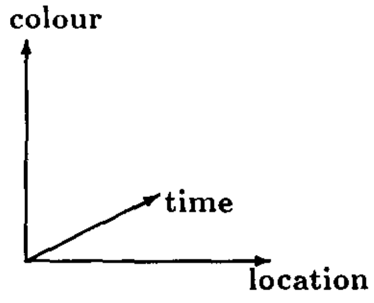

Figure 1: Three typical dimensions: location, colour and time.

Several authors have extended the basic Petri net model with *coloured* or *typed tokens* ([1, 5, 6, 8]). In these models tokens have a value, often referred to as 'colour'. There are several reasons for such an extension. One of these reasons is the fact that (uncoloured) Petri nets tend to become too large to handle. Another reason is the fact that tokens often represent objects or resources in the modelled system. As such, these objects may have attributes, which are not easily represented by a simple Petri net token.
Other authors have proposed a Petri net model with explicit quantitative time (e.g. [1, 3, 6, 10, 16]). We call these models *timed Petri net* models.
Other extensions are the introduction of token identifications ([7]), tokens having a life-cycle, priorities, fuzzy Petri nets ([4]), etc.

Practical experiences show that at least some of these extensions are necessary to make the Petri net model suitable for the modelling of large and complex systems. In this paper we introduce the **multi-dimensional Petri net** model, which is an attempt to generalize these extensions in a unifying way.
In a multi-dimensional Petri net there are **dimensions** instead of places. In an n-dimensional Petri net, each token has n dimensions. Typical dimensions are the 'place dimension' (location), the 'colour dimension' and the 'time dimension', see figure 1. However, many other dimensions are possible, e.g. a 'space dimension', a 'weight dimension', a 'financial dimensions', etc. For example, if we want to model fuzzy information, we can add a 'truth dimension' that specifies the truth value of each token. (The truth value is a value between 0 (false) and 1 (true), see [4].) In this paper we will show that we can use a multi-dimensional Petri net to model all of these aspects in a unifying way. (Nevertheless, there are some extensions which do not fit directly the framework presented in this paper, e.g. hierarchical Petri nets.)

Compared to other Petri net models there are two remarkable differences: the introduction of dimensions and the absence of places. To be able to use the modelling and analysis techniques developed for other Petri net models, we will investigate the relations between multi-dimensional Petri nets and classical Petri nets, coloured

Petri nets and timed Petri nets.

When analysing a system modelled in terms of a multi-dimensional Petri net, we are often interested in properties and/or performance characteristics that relate to a *restricted* set of dimensions. This is the reason we define a new concept called **projection**. A multi-dimensional Petri net (MDPN) can be projected onto a restricted set of dimensions. The projected MDPN can be used to answer certain questions about the original MDPN. This approach is often favourable since analysis of the projected net tends to be less complicated.
Consider for example a MDPN with two dimensions: a space dimension and a time dimension. We can investigate whether some critical position in the space dimension is reachable by analysing the MDPN projected onto the space dimension.

The remainder of this paper consists of 6 sections. Section 2 introduces the multi-dimensional Petri net model. In section 3 we give an example of a multi-dimensional Petri net which describes a warehouse. Section 4 discusses the relation with other net models and section 5 gives a brief survey of existing analysis methods. In section 6 we introduce the concept of projection and prove that it can be used to analyse a multi-dimensional Petri net more efficiently. In section 7, we finish with a conclusion.

2 Multi-dimensional Petri nets

In this section we present the multi-dimensional Petri net model. We use a small example to introduce some of the concepts, followed by a formal definition and semantics.

2.1 Informal introduction

To introduce the multi-dimensional Petri net model, we will use an example informally described in terms of a coloured Petri net. This way we also indicate how other kinds of Petri nets (e.g. coloured Petri nets) can be transformed into a multi-dimensional Petri net. (The relation between the multi-dimensional Petri net model and other Petri net based models is discussed in section 4.)
Figure 2 shows a coloured Petri net composed of five places (`waiting`, `in_progress`, `ready`, `free` and `busy`) and three transitions (`start`, `finish` and `prepare`). There are two kinds of tokens: tokens that represent tasks and tokens that represent resources. Tokens in the places `waiting`, `in_progress` and `ready` represent tasks (e.g. jobs). Tokens in the places `free` and `busy` represent resources (e.g. machines). Resources can be used to execute tasks. For each resource it is specified which tasks it can perform. In this coloured Petri net, tokens have a value (colour). A task is represented by a token whose value is a capital letter. A resource is represented by a token whose value is a set of capitals, indicating which tasks the resource can perform.

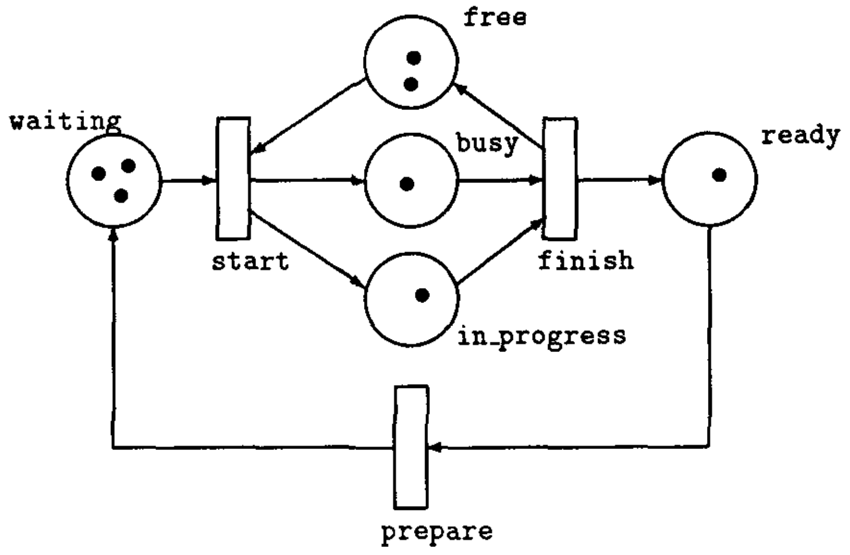

Figure 2: A coloured Petri net.

Transition **start** may fire if it is able to consume a task from place **waiting** and a resource from place **free**. If **start** fires, the task and the resource are removed from the input places **waiting** and **free**, and put in the places **in_progress** and **busy** respectively. Transition **finish** releases the resource and puts the token consumed from **in_progress** into place **ready**. Transition **prepare** transforms finished tasks into new tasks.

Now we show how we can model this example in terms of a **multi-dimensional Petri net**. In the multi-dimensional Petri net there are *no places*, only **dimensions**. In this example there are two dimensions: the 'place dimension' D_{place} and the 'colour dimension' D_{colour}. The place dimension specifies the location of a token, i.e. the place in which the token resides:

$$D_{place} = \{ \texttt{waiting}, \texttt{in\_progress}, \texttt{ready}, \texttt{free}, \texttt{busy} \}$$

The colour dimension specifies the value (colour) of a token:[1]

$$\begin{aligned} Task &= \{'A','B',..,'Z'\} \\ Resource &= \mathbb{P}(Task) \\ D_{colour} &= Task \cup Resource \end{aligned}$$

Since there are two dimensions: each **token** is represented by a pair $\langle p, c \rangle$ where $p \in D_{place}$ and $c \in D_{colour}$. The **token domain** D is the set of all possible pairs, i.e. $D = D_{place} \times D_{colour}$. The **state** s of a multi-dimensional Petri net is a *multi-set* of tokens, i.e. $s \in D_{MS}$. In appendix A.1 a short introduction to multi-sets, also referred to as bags, is given.

[1] $\mathbb{P}(A)$ is the powerset of A.

In the multi-dimensional Petri net there are three transitions:

$$T = \{ \texttt{start}, \texttt{finish}, \texttt{prepare} \}$$

Each of these transitions corresponds to one transition in the coloured Petri net. The **transition relation** F describes for each transition the relation between the multi-set of consumed tokens and the multi-set of produced tokens. Consider for example the transition prepare:

$$F_{\texttt{prepare}} = \{ (\underbrace{\langle \texttt{ready}, c \rangle}_{\text{consume}} , \underbrace{\langle \texttt{waiting}, c \rangle}_{\text{produce}}) \mid c \in \mathit{Task} \}$$

This relation is a set of pairs. Each pair specifies a *possible* consumption and production, i.e. the pair $(\langle \texttt{ready}, c \rangle , \langle \texttt{waiting}, c \rangle)$ says that transition **prepare** is allowed to produce a token $\langle \texttt{waiting}, c \rangle$ if it consumes a token $\langle \texttt{ready}, c \rangle$. Note that $\langle \texttt{waiting}, c \rangle$ and $\langle \texttt{ready}, c \rangle$ are both multi-sets containing precisely one element (see appendix A.1).
A transition is called **enabled** if there are 'enough' tokens of the 'right' type. In this particular example, transition prepare is enabled if there is at least one token represented by $\langle \texttt{ready}, c \rangle$ where c is an arbitrary capital. An enabled transition may **fire (occur)**. Firing a transition means consuming and/or producing tokens as specified by the transition relation. If prepare fires, it consumes a token $\langle \texttt{ready}, c \rangle$ and produces a token $\langle \texttt{waiting}, c \rangle$, i.e. a task c is transferred from stage **ready** to stage **waiting**.
A more complex transition is the transition start:

$$F_{\texttt{start}} = \{ (\overbrace{\langle \texttt{waiting}, c \rangle + \langle \texttt{free}, r \rangle}^{\text{consume}} , \overbrace{\langle \texttt{in\_progress}, c \rangle + \langle \texttt{busy}, r \rangle}^{\text{produce}}) \mid c \in \mathit{Task} \ \wedge \ r \in \mathit{Resource} \ \wedge \ c \in r \}$$

Transition **start** is enabled if there are two tokens $\langle \texttt{waiting}, c \rangle$ and $\langle \texttt{free}, r \rangle$ such that $c \in r$, i.e. task c can be performed by resource r. If transition **start** fires while consuming the multi-set $\langle \texttt{waiting}, c \rangle + \langle \texttt{free}, r \rangle$, then the two tokens specified by the multi-set $\langle \texttt{in\_progress}, c \rangle + \langle \texttt{busy}, r \rangle$ are produced.
Finally, the transition **finish** is specified as follows:

$$F_{\texttt{finish}} = \{ (\overbrace{\langle \texttt{in\_progress}, c \rangle + \langle \texttt{busy}, r \rangle}^{\text{consume}} , \overbrace{\langle \texttt{ready}, c \rangle + \langle \texttt{free}, r \rangle}^{\text{produce}}) \mid c \in \mathit{Task} \ \wedge \ r \in \mathit{Resource} \ \wedge \ c \in r \}$$

Transition **finish** releases the resource and transfers a task from stage **in_progress** to stage **ready**.

In this example there are two dimensions. However, it is also possible to have multi-dimensional Petri net with a 'time dimension', an extra 'space dimension', etc.

Another example of a possible dimension is the 'identification dimension', i.e. each token has some identification. This concept agrees with the object-oriented approach. Tokens represent objects and objects have a unique identity. This is quite natural: though properties of an object may change in time, the object as it is does not change. These token identities can also be used to relate objects, e.g. in the example just given we can add token identities and use them to specify that transition `finish` only consumes tasks and resources that are related. In Van Hee and Verkoulen [7] it is shown how to calculate unique token identifiers.
In fuzzy Petri nets (Chen, Ke and Chang [4]), tokens have a 'truth value' and transitions have a 'certainty factor'. The truth value of a token may be any real number between between 0 (false) and 1 (true). This can be modelled by adding a 'truth value dimension'.

Since there are no explicit places in a multi-dimensional Petri net, there is no straightforward graphical notation (especially when there are many dimensions). However, it is really easy to transform a multi-dimensional Petri net into other Petri nets (e.g. a coloured Petri net) and vice versa (see section 4).

2.2 Formal definition

In this subsection we define multi-dimensional Petri nets in mathematical terms, such as functions, multi-sets and relations.

Definition 1 (MDPN)
A **multi-dimensional Petri net** is a tuple MDPN $= (D_1, D_2, .., D_n, T, F)$ satisfying the following requirements:

(i) $n \in \mathbb{N}$ is the number of dimensions.

(ii) $D_1, D_2, ..D_n$ are finite sets, the **dimensions**.

(iii) $D = D_1 \times D_2 \times .. \times D_n$, the **token domain**.

(iv) T is a finite set of **transitions**.

(v) Function F specifies the **transition relation** of each transition. F is defined from T into relations. If $t \in T$, then:

$$F_t \subseteq D_{MS} \times D_{MS} \quad \text{and} \quad \#F_t \text{ is finite}$$

Instead of places there are dimensions. Each dimension refers to a specific aspect of a token. Typical dimensions are the space or location dimension, the time dimension and the colour or value dimension. The number of dimensions n is a variable, i.e. n may vary from application to application. A token is represented by an element of the token domain D, i.e. in an n-dimensional Petri net a token is described by an n-tuple $\langle d_1, d_2, .., d_n \rangle$

Each transition is specified by the transition relation ($dom(F) = T$). Let t be a transition, i.e. $t \in T$. F_t is a finite set of pairs $\langle b_{in}, b_{out}\rangle$, where b_{in} and b_{out} are both multi-sets over D. If $\langle b_{in}, b_{out}\rangle \in F_t$, then transition t is enabled is state s, if at least the tokens specified by b_{in} are present in s. If this is the case, then t may consume the tokens specified by b_{in} and produce the tokens specified by b_{out}, i.e t is allowed to fire.
The restrictions that each dimension is finite (ii) and that each F_t has to be a finite set (v) are used in section 4. These restrictions have been added to be able to construct an equivalent classical Petri net.

2.3 Semantics of multi-dimensional Petri nets

The tuple $(D_1, D_2, ..D_n, T, F)$ specifies the static structure of an MDPN. In the remainder of this section we define the behaviour of an multi-dimensional Petri net, i.e. the semantics of the MDPN model.

Definition 2
A **state** is defined as a multi-set of tokens. S is the **state space**, i.e. the set of all possible states:

$$S = D_{MS}$$

The state of the a MDPN is a multi-set of tokens, i.e. a multi-set of n-tuples. Possible states of the MDPN described in section 2.1 are: $\emptyset_D$, $\langle \texttt{ready}, \{'A', 'B'\}\rangle$, $\langle \texttt{waiting}, 'C'\rangle + 2\langle \texttt{free}, \emptyset\rangle, 5\langle \texttt{ready}, 'Z'\rangle + 3\langle \texttt{ready}, 'N'\rangle + \langle \texttt{busy}, \{'A', 'B'\}\rangle$. Note that these states contain 0, 1, 3 and 9 tokens respectively.

Definition 3
An **event** is a triple $\langle t, b_{in}, b_{out}\rangle$, which represents the possible firing of transition t while removing the tokens specified by the multi-set b_{in} and adding the tokens specified by the multi-set b_{out}. E is the **event set:**

$$E = T \times D_{MS} \times D_{MS}$$

An event $e = \langle t, b_{in}, b_{out}\rangle$ represents the firing of t while consuming the tokens specified by b_{in} and producing the tokens specified by b_{out}.
An event is enabled in state s iff (i) the tokens to be consumed are present in s and (ii) an 'allowed' collection of tokens is to be produced (as specified by the transition relation).

Definition 4
An event $\langle t, b_{in}, b_{out} \rangle \in E$ is **enabled** in state $s \in S$ iff:

(i) $b_{in} \leq s$

(ii) $\langle b_{in}, b_{out} \rangle \in F_t$

Consider the MDPN described in section 2.1. Event:

$$\langle \texttt{start} , \langle \texttt{waiting}, 'A' \rangle + \langle \texttt{free}, \{'A', 'B'\} \rangle , \langle \texttt{in\_progress}, 'A' \rangle + \langle \texttt{busy}, \{'A', 'B'\} \rangle \rangle$$

is enabled in state:

$$2\langle \texttt{waiting}, 'A' \rangle + \langle \texttt{waiting}, 'B' \rangle + 3\langle \texttt{free}, \{'A', 'B'\} \rangle$$

Event:

$$\langle \texttt{finish} , \langle \texttt{in\_progress}, 'A' \rangle + \langle \texttt{busy}, \{'A', 'B'\} \rangle , \langle \texttt{ready}, 'B' \rangle + \langle \texttt{free}, \{'A', 'B'\} \rangle \rangle$$

is not enabled in state:

$$2\langle \texttt{waiting}, 'A' \rangle + \langle \texttt{waiting}, 'B' \rangle + 3\langle \texttt{free}, \{'A', 'B'\} \rangle$$

because both conditions (i) and (ii) are violated.

Definition 5
When an event $\langle t, b_{in}, b_{out} \rangle$ is enabled in state s_1, it may **occur**, i.e. transition t **fires** while removing the tokens specified by b_{in} and adding the tokens specified by b_{out}.
If $\langle t, b_{in}, b_{out} \rangle$ occurs in state s_1, then the net changes into the state s_2, defined by:

$$s_2 = (s_1 - b_{in}) + b_{out}$$

State s_2 is said to be **directly reachable** from s_1 by the occurrence of event $e = \langle t, b_{in}, b_{out} \rangle$, this is also denoted by:

$$s_1 \xrightarrow{e} s_2$$

Moreover, $s_1 \longrightarrow s_2$ means that there exists an enabled event e such that $s_1 \xrightarrow{e} s_2$.

Definition 6
A **firing sequence** is a sequence of states and events:

$$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} s_4 \xrightarrow{e_4} ..$$

State s_n is **reachable** from s_1 iff there exists a firing sequence of finite length starting in s_1 and ending in s_n:

$$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} .. \xrightarrow{e_{n-1}} s_n$$

This concludes the definition and semantics of the MDPN model. Compared to

other Petri net models there are two remarkable differences: (1) the introduction of dimensions and (2) the absence of places. In fact, the place where the token resides can be seen as just another dimension (the 'place dimension', see section 2.1). The concept of dimensions allows us to deal with extensions like token colours, token identifications, time, etc. in a unifying way.
However, to be able to use existing modelling and analysis methods, we have to establish formal relationships with other net models. This will be discussed in section 4. Finally, in section 6 we will focus on one of the merits of multi-dimensional Petri nets: the ability to define and use projections.

3 An example

To illustrate the MDPN model, we give a fictitious example of a warehouse modelled in terms of a multi-dimensional Petri net. Consider a warehouse divided into a number of sections. Each section is characterised by an X-coordinate and a Y-coordinate. The warehouse stores goods that are stacked on pallets. A pallet is either used or it is empty. Pallets can be transported from one section to another by a forklift truck. To move goods from section $s1$ to section $s2$, an unloaded forklift moves to section $s1$, loads the corresponding pallet and transports it to section $s2$ where it is unloaded. Figure 3 shows a schematic representation of the warehouse.

In this particular case we identify 5 dimensions:

$$\begin{aligned}
D_{kind} &= \{\texttt{pallet}, \texttt{forklift}\} \\
D_X &= \{1,2,3,4,5,6,7,8,9,10\} \\
D_Y &= \{1,2,3,4,5,6,7,8,9,10\} \\
D_{state} &= \{\texttt{free}, \texttt{busy}\} \\
D_{contents} &= \mathbb{P}(\{'A','B',..'Z'\})
\end{aligned}$$

The *kind* dimension says whether an token represents a pallet or a forklift. The X and Y dimensions specify the whereabouts of a pallet or a forklift. The *state* dimension is used to indicate the fact that a pallet is used or a forklift is loaded. The contents of a pallet or the forklift carrying a pallet is given by the *contents* dimension. Note that the goods on a pallet are represented by a set of product identifiers, i.e. a subset of $\{'A','B',..'Z'\}$.
An example of an element of token domain D is $\langle \texttt{pallet}, 3, 4, \mathit{free}, \emptyset \rangle$, this is an empty pallet located in section $(3,4)$.

There are three transitions,

$$T = \{\texttt{load}, \texttt{unload}, \texttt{move}\}$$

The transition `load` represents the activity of loading a pallet by some forklift truck. The transition relation for `load` is defined as follows:

$$F_{\texttt{load}} = \{ (\langle \texttt{forklift}, x, y, \texttt{free}, \emptyset \rangle + \langle \texttt{pallet}, x, y, s, c \rangle \ ,$$

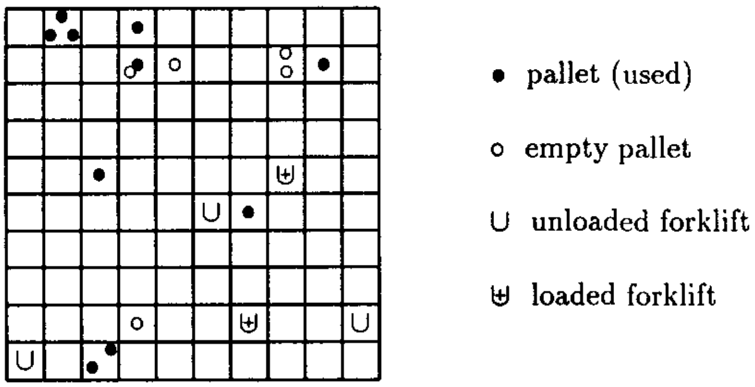

Figure 3: The warehouse.

$$\begin{aligned} &\langle \texttt{forklift}, x, y, \texttt{busy}, c\rangle\ \rangle \\ &\mid x \in D_X \ \wedge\ y \in D_Y \ \wedge\ s \in D_{state} \ \wedge\ c \in D_{contents}\} \end{aligned}$$

Note that the forklift has to be unloaded to be able to load a pallet and that the pallet and the forklift have to be in the same section.

Transition `unload` fires when a pallet is unloaded by a forklift. The transition relation for `unload` is defined as follows:

$$\begin{aligned} F_{\texttt{unload}} \ = \ \{\langle\ &\langle \texttt{forklift}, x, y, \texttt{busy}, c\rangle\ , \\ &\langle \texttt{forklift}, x, y, \texttt{free}, \emptyset\rangle + \langle \texttt{pallet}, x, y, s, c\rangle\ \rangle \\ &\mid x \in D_X \ \wedge\ y \in D_Y \ \wedge\ s \in D_{state} \ \wedge\ c \in D_{contents} \ \wedge \\ &\quad ((s = \texttt{free}) \Leftrightarrow (c = \emptyset))\} \end{aligned}$$

Only loaded forklifts can be unloaded.

Transition `move` models the actual transportation of a pallet, but also all other movements of a forklift:

$$\begin{aligned} F_{\texttt{move}} \ = \ \{\langle\ &\langle \texttt{forklift}, x, y, s, c\rangle\ , \\ &\langle \texttt{forklift}, x', y', s, c\rangle\ \rangle \\ &\mid x, x' \in D_X \ \wedge\ y, y' \in D_Y \ \wedge\ s \in D_{state} \ \wedge\ c \in D_{contents} \ \wedge \\ &\quad (-1 \leq x - x' \leq 1) \ \wedge\ (-1 \leq y - y' \leq 1)\} \end{aligned}$$

Note that only movements between adjacent sections are allowed.

This completes the definition of the multi-dimensional Petri net $(D_{kind}, D_X, D_Y, D_{state}, D_{contents}, T, F)$ which models the warehouse. This example will be used to illustrate some of the concepts introduced in the remainder.

4 Relations with other net models

In this section we reveal some of the relationships with other Petri net based models. These inter-relationships are very important, since we want to apply the rich set of modelling and analysis techniques developed for other Petri net models, e.g. timed and/or coloured Petri net models.
First we discuss the relation between the classical Petri net model and the multi-dimensional Petri net model. To do this we need a definition of the classical Petri net. (Similar definitions are given in [12, 13, 15].)

Definition 7 (Petri net)
A classical Petri net is a three tuple PN $= (P, T, F)$, where:

(i) P is a finite set of **places**.

(ii) T is a finite set of **transitions**.

(iii) $F \in (P \times T)_{MS} \cup (T \times P)_{MS}$ specifies the **flow relation**, i.e. the arcs connecting places and transitions.

The flow relation F describes how the places and transitions are connected. If $F(\langle p,t\rangle) = k$, then there are k parallel arcs from place p to transition t, i.e. if $k > 0$ then p is an input place of t. If $F(\langle t,p\rangle) = k$, then there are k parallel arcs from transition t to place p, i.e. if $k > 0$ then p is an output place of t.
The state of a Petri net, often referred to as marking, is a multi-set of places. A transition in a classical Petri net is enabled if each input place contains 'sufficient' tokens. An enabled transition may fire. If a transition t fires, the specified number of tokens are consumed from the input places and the specified number of tokens are produced for the output places.

Now we are able to express any classical Petri net in terms of a MDPN. Since there are identifiers (e.g. T and F) that are used in both models, all identifiers referring to the MDPN model are superscripted by a horizontal line (this to avoid confusion).

Theorem 1 (PN $\rightarrow$ MDPN)
Given a classical Petri net PN $= (P, T, F)$, we can construct an equivalent multi-dimensional Petri net MDPN $= (\overline{D}_1, \overline{T}, \overline{F})$ as follows:

(i) $n = 1$

(ii) $\overline{D}_1 = P$

(iii) $\overline{T} = T$

(iv) For all $t \in T$, we define:

$$\overline{F}_t = \{\langle \sum_{p \in P} F(p,t)p, \sum_{p \in P} F(t,p)p \rangle\}$$

The constructed MDPN has only one dimension P. Each transition in the classical Petri net corresponds to precisely one transition in the constructed MDPN. For each transition $t \in T$, the relation $\overline{F}_t$ contains only one pair. The first element $\sum_{p \in P} F(p,t)p$ represents the bag of input places of t, the second element $\sum_{p \in P} F(t,p)p$ corresponds to the bag of output places of t.
We define the nets to be **equivalent** if and only if there is a bijection f between two state spaces such that $s_1 \longrightarrow s_2$ in one net if and only if $f(s_1) \longrightarrow f(s_2)$ in the other net and vice versa. It is easy to verify that these nets are equivalent.
The opposite, i.e. the construction of an equivalent classical Petri net, is also possible because of the requirement that the domain $\overline{D}$ of the MDPN is finite.

Theorem 2 (MDPN $\rightarrow$ PN)
Given a multi-dimensional Petri net MDPN $= (\overline{D}_1, \overline{D}_2, .., \overline{D}_n, \overline{T}, \overline{F})$, we can construct an equivalent classical Petri net PN $= (P, T, F)$ as follows:

(i) $P = \overline{D} = \overline{D}_1 \times \overline{D}_2 \times .. \times \overline{D}_n$

(ii) $T = \{\langle t, b_{in}, b_{out}\rangle \in \overline{E} \mid \langle b_{in}, b_{out}\rangle \in \overline{F}_t\}$

(iii) The flow relation:

$$F = \sum_{t' = \langle t, b_{in}, b_{out}\rangle \in T} \sum_{p \in P} b_{in}(p)\langle p, t'\rangle \ + \ b_{out}(p)\langle t', p\rangle$$

This transformation requires some more explanation. Every element in the token domain $\overline{D}$ corresponds to precisely one place in the classical Petri net. This is the reason the token domain has to be finite. Furthermore, every event which can be enabled in some state of the MDPN corresponds to a unique transition in the classical Petri net. If $t' = \langle t, b_{in}, b_{out}\rangle$ is such an event, then t' consumes tokens from the input places specified by b_{in} and produces tokens for the output places specified by b_{out}. This information is used to construct the flow relation F.
We will use the example given in section 3 to clarify this construction. If we trans-

form the multi-dimensional Petri net $(D_{kind}, D_X, D_Y, D_{state}, D_{contents}, T, F)$ into a classical Petri net PN, then PN contains one place for each element in the token domain and one transition for each possible event. The classical Petri net PN contains $2 \times 10 \times 10 \times 2 \times 2^{26} \approx 2.68 \times 10^{10}$ places. Transition `load` corresponds to $10 \times 10 \times 2 \times 2^{26}$ transitions in PN and transition `unload` corresponds to $10 \times 10 \times 2^{26}$ transitions. Transition `move` is transformed into $(2+8\times 3+2)^2 \times 2 \times 2^{26}$ transitions in PN. Thus, the classical Petri net contains approximately 1.25×10^{11} transitions.

Note that the two constructions given in this section 'circular'. If we construct a multi-dimensional Petri net MDPN for a classical Petri net PN1 and we construct another classical Petri net PN2 for this MDPN, then PN1 and PN2 are equivalent. If we construct a classical Petri net PN for a multi-dimensional Petri net MDPN1 and we construct another multi-dimensional Petri net MDPN2 for this PN, then MDPN1 and MDPN2 are also equivalent.

It is also possible to transform a coloured Petri net into an equivalent multi-dimensional Petri net and vice versa. This subject is discussed in appendix A.2.
In appendix A.3 we show how we can handle time in a multi-dimensional Petri net.

These transformations are quite straightforward. The simplicity of these transformations promises a smooth transition from a classical, coloured and/or timed Petri net to a multi-dimensional Petri net and vice versa. This way it is possible to use parts of the theory, methods and tools developed for other Petri net based models.

5 Analysis of multi-dimensional Petri nets

There are several reasons for modelling a system, e.g. to create and evaluate a design of a new system, to compare alternative designs and to investigate possible improvements in a real system. Model building forces us to organize, evaluate and examine the validity of our thoughts. This way modelling reveals errors and possible improvements.
In essence, the modelling process serves two purposes. First of all, the model is used as a 'blueprint' of the system under consideration, e.g. the design of a new system or a plan which describes improvements. Secondly, models are used to analyse certain aspects of a system, e.g. the performance, efficiency or correctness of a system. Since analysis is often the main goal of model building, we have to supply suitable analysis methods.
Multi-dimensional Petri nets will *not* provide new analysis techniques that are not feasible for classical Petri nets, coloured Petri nets or timed Petri nets. However, most of the existing analysis methods can be applied to multi-dimensional Petri nets. This is the reason we start with a brief survey of existing Petri net based analysis techniques.

A lot of analysis techniques have been developed in the area of pure Petri net theory.

Most of them are based on the classical Petri net model.
Many of these techniques have been extended to analyse high-level Petri nets, for example reachability graphs and invariants. Recall that as long as the number of colours is finite, a high-level net can be 'unfolded' into an equivalent, but much larger, Petri net without colours. The unfolding of nets has been studied to see how the analysis methods for high-level nets should work. For the moment, however, it is only possible to use these methods for relatively small systems and for selected parts of larger systems.
An example of such a method is the creation of a *reachability graph* for high-level nets. Because of the explosion of the number of states, these graphs tend to become too large to analyse. Several reduction techniques have been proposed to deal with this problem. None of them gives a satisfactory solution (see Jensen [8]).
Another analysis technique available for high-level Petri nets is the generation of *place and transition invariants.* These invariants are used to derive and prove properties of the modelled system. A place invariant (P-invariant) is a weighted token sum, i.e. a weight is associated with every token in the net. This weight is based on the location (place) and the value (colour) of the token. A place invariant holds if the weighted token sum of all tokens remains constant during the execution of the net. Transition invariants (T-invariants) are the duals of place invariants and the basic idea behind them is to find firing sequences with no effects, i.e. firing sequences which reproduce the initial state. Some analysis techniques have been developed to calculate these invariants automatically (see Jensen [8]). These techniques have a number of problems. For large nets with a lot of different colours, it is hard to compute these invariants. Usually there are infinitely many invariants (a linear combination of invariants is also an invariant), therefore it is difficult to distill the interesting ones. However, there is a more promising way to use invariants. If the user supplies a number of invariants, it is easy to verify these invariants totally automatically. If an invariant does not hold, it is relatively easy to see how the Petri net (or the invariant) should be modified. The latter approach does not solve the problem that applying invariants requires a lot of training.

The addition of time to the classical Petri net model resulted in a lot of new and interesting techniques to analyse the dynamic behaviour of a system. Literature on this subject reflects the fact that the study of timed Petri nets developed along two separate lines.
The first line concentrates on the verification of dynamic properties. Most of the methods developed along this line are based on nets with deterministic delays. A serious drawback of these methods is the fact that in many real systems the activity durations are not fixed, i.e. they vary because of disturbances and other interferences. Assuming deterministic delays often results in inaccurate results.
The second line concentrates on the performance evaluation of timed Petri nets by means of analysis of the underlying stochastic process. Instead of assuming deterministic activity durations, an attempt is made to capture the essence of a system by probabilistic assumptions. These probabilistic assumptions often include the distribution of the delays in the net. For analysis reasons, these distributions are

assumed to be negatively exponential. Molloy showed that, due to the memoryless property of the exponential distribution, such a stochastic TPN is isomorphic to a continuous time Markov chain ([11]). This allows for analytical methods to analyse the dynamic behaviour of a system, this way it is possible to calculate performance measures, e.g. the average waiting time or the probability of having more than five tokens in a specific place. An example of a stochastic TPN model is the Generalized Stochastic Petri Net (GSPN) model developed by Ajmone Marsan et al. ([10]). Many authors give conditions for the topology of the net or the distribution of the delays such that analysis of the underlying stochastic process is possible. In general these conditions are quite strong. Moreover, for real problems, the state space of the corresponding continuous time Markov chain tends to be too large to analyse.

A small number of analysis methods have been presented for Petri nets with interval timing ([1, 3]). These methods can be used to calculate *bounds* for all kinds of performance measures (e.g. response times, waiting times and utilization). Note that these bounds are *safe*, i.e. it is possible to prove their correctness.

The analysis techniques discussed so far can be used for the verification of specific properties. Validation requires a different approach. Instead of giving a formal proof a number of experiments are conducted to test hypotheses or to estimate certain performance measures. In general validation is done by means of *simulation.*

In section 4 we demonstrated that a smooth transition from a MDPN to another Petri net based model is possible. Therefore, most of the techniques developed for classical Petri nets, coloured Petri nets and timed Petri nets can be applied to multi-dimensional Petri nets.

As an example, we focus on the so-called place invariants:

Definition 8 (Invariant)
Let MDPN $= (D_1, D_2, ..D_n, T, F)$ be a multi-dimensional Petri net and $w \in D \to \mathbb{Z}$ is a **weight function** which assigns a weight to every token in the net. This weight function w is an **invariant** if and only if for each event $e = \langle t, b_{in}, b_{out}\rangle$ with $\langle b_{in}, b_{out}\rangle \in F_t$ the following statement holds:

$$\sum_{d \in D} w(d) b_{in}(d) \quad = \quad \sum_{d \in D} w(d) b_{out}(d)$$

Similar definitions of place invariants given in [13, 15, 9].

Consider for example the net given in section 2.1. Let $w_1 \in D \to \mathbb{Z}$ be the weight function which assigns weight 1 to the tokens residing in one of the the following places: `waiting`, `in_progress` or `ready`, i.e. for $c \in D_{colour}$, we have:

$$w_1(\langle \texttt{waiting}, c\rangle) = w_1(\langle \texttt{in\_progress}, c\rangle) = w_1(\langle \texttt{ready}, c\rangle) = 1$$
$$w_1(\langle \texttt{free}, c\rangle) = w_1(\langle \texttt{busy}, c\rangle) = 0$$

This weight function is an invariant which specifies that the number of tasks is constant.
If w_2 is such that for any $c \in D_{colour}$:

$$w_2(\langle \texttt{waiting}, c\rangle) = w_2(\langle \texttt{in\_progress}, c\rangle) = w_2(\langle \texttt{ready}, c\rangle) = 0$$
$$w_2(\langle \texttt{free}, c\rangle) = w_2(\langle \texttt{busy}, c\rangle) = 1$$

Then w_2 is an invariant which specifies that the number of resources is constant.

In the warehouse modelled in section 3, the number of forklift trucks is constant. Let w_3 be such that for any $d \in D$:[2]

$$w_3(d) = \begin{cases} 1 & \text{if } \pi_{kind}(d) = \texttt{forklift} \\ 0 & \text{if } \pi_{kind}(d) = \texttt{pallet} \end{cases}$$

Weight function w_3 is invariant, i.e. there is conservation of forklifts. We can also use an invariant w_4 to prove that the number of goods is constant, i.e. goods cannot get 'lost'.

Suppose one of the dimensions of a multi-dimensional Petri net is the 'time dimension'. In this case it may be possible to prove temporal properties by calculating the invariants of the multi-dimensional Petri net. Note that in most Petri net models extended with time and colour, time aspects are abstracted from before invariants are calculated/generated.

We are often only interested in a subset of the dimensions of a multi-dimensional Petri net, i.e. the questions we want to answer relate to a limited number of dimensions. Consider for example the invariant w_3, D_{kind} is the only dimension w_3 has a bearing on. Invariants w_1 and w_2 only relate to the *place* dimension. It seems inefficient to analyse the entire multi-dimensional Petri net if we are only interested in a few dimensions. Therefore, we will focus on projections of a MDPN.

6 Projections

In this section we introduce a new concept, called 'projection'. If we project a MDPN onto a restricted number of dimensions, we can abstract from the dimensions we are not interested in. For example, if we want to prove the conservation of forklift trucks and goods in a warehouse, then it suffices to analyse the MDPN projected onto the *kind* and *contents* dimensions. We will formally prove that it is possible to analyse certain aspects of a MDPN by analysing the proper projection of the MDPN. Analysis of a projected MDPN is often much more efficient, since the token domain used by the projected MDPN is smaller.

[2] π_{kind} is the value of the *kind* field of tuple d.

To be able to define the projection concept for multi-dimensional Petri nets, we need to define projection of tuples and states.

Definition 9
Let MDPN $= (D_1, D_2, ..D_n, T, F)$ be a multi-dimensional Petri net, $D = D_1 \times D_2 \times .. \times D_n$, $d = \langle d_1, d_2, .., d_n \rangle$, $d \in D$, $S = D_{MS}$ and $s \in S$.
For $i \in \{1, 2, .., n\}$, we define:

$$\begin{aligned} \pi_i(d) &= d_i \\ \prod_i(s) &= \sum_{d \in s} s(d)\pi_i(d) \end{aligned}$$

For $I = \{i_1, i_2, .., i_m\}$ such that $I \subseteq \{1, 2, .., n\}$ and $i_1 < i_2 < .. < i_m$, we define:

$$\begin{aligned} \pi_I(d) &= \langle d_{i_1}, d_{i_2}, .., d_{i_m} \rangle \\ \prod_I(s) &= \sum_{d \in s} s(d)\pi_I(d) \end{aligned}$$

The projection operator π_I is used to project a tuple onto a selected set of dimensions as specified by I. The projection operator $\prod_I$ is used to project a state onto a selected set of dimensions. Consider for example state s in the MDPN described in section 2.1:

$$s = 2\langle \texttt{waiting}, 'A' \rangle + \langle \texttt{waiting}, 'B' \rangle + 3\langle \texttt{free}, \{'A', 'B'\} \rangle$$

If we project s onto the place dimension, then $I = \{place\}$ and:

$$\prod_I(s) = 3\ \texttt{waiting} + 3\ \texttt{free}$$

Definition 10 (Projection)
Let MDPN $= (D_1, D_2, ..D_n, T, F)$ be a multi-dimensional Petri net and $I = \{i_1, i_2, .., i_m\}$ such that $I \subseteq \{1, 2, .., n\}$ and $i_1 < i_2 < .. < i_m$.
The **I-projection** of MDPN is denoted by $\prod_I(\text{MDPN})$ and $\prod_I(\text{MDPN}) = (D_{i_1}, D_{i_2}, ..D_{i_m}, T, \overline{F})$ where for each $t \in T$:

$$\overline{F}_t = \{\langle \prod_I(b_{in}), \prod_I(b_{out}) \rangle \mid \langle b_{in}, b_{out} \rangle \in F_t\}$$

Let MDPN $= (D_{place}, D_{colour}, T, F)$ be the multi-dimensional Petri net defined in section 2.1 and $I_1 = \{place\}$. The I_1-projection of MDPN is the three tuple $(D_{place}, T, \overline{F})$ with:

$$\begin{aligned} D_{place} &= \{\ \texttt{waiting}, \texttt{in\_progress}, \texttt{ready}, \texttt{free}, \texttt{busy}\ \} \\ T &= \{\ \texttt{start}, \texttt{finish}, \texttt{prepare}\ \} \\ \overline{F}_{\texttt{prepare}} &= \{\langle \texttt{ready}, \texttt{waiting} \rangle\} \\ \overline{F}_{\texttt{start}} &= \{\langle \texttt{waiting} + \texttt{free}, \texttt{in\_progress} + \texttt{busy} \rangle\} \\ \overline{F}_{\texttt{finish}} &= \{\langle \texttt{in\_progress} + \texttt{busy}, \texttt{ready} + \texttt{free} \rangle\} \end{aligned}$$

$$\begin{tikzcd}
\textbf{MDPN} \arrow[r, "projection"] \arrow[d, dashed] & \Pi_I(\textbf{MDPN}) \arrow[d, "analysis"] \\
\textbf{results} & \textbf{results} \arrow[l, "interpretation"']
\end{tikzcd}$$

Figure 4: Analysis of a MDPN by analysing a projection of the MDPN.

In this example we abstracted from the token colours. It is also possible to abstract from the place dimension, i.e. projecting the MDPN onto the colour dimension. The I_2-projection of MDPN with $I_2 = \{colour\}$ is the tuple $(D_{colour}, T, \overline{F})$ with:

$$\begin{aligned}
D_{colour} &= Task \cup Resource \\
T &= \{\ \texttt{start}, \texttt{finish}, \texttt{prepare}\ \} \\
\overline{F}_{\texttt{prepare}} &= \{\langle c, c\rangle \mid c \in Task\} \\
\overline{F}_{\texttt{start}} &= \{\langle c+r, c+r\rangle \mid c \in Task\ \wedge\ r \in Resource\ \wedge\ c \in r\} \\
\overline{F}_{\texttt{finish}} &= \{\langle c+r, c+r\rangle \mid c \in Task\ \wedge\ r \in Resource\}
\end{aligned}$$

These projections may be interesting from an analysis point of view: it is possible to obtain analysis results for a multi-dimensional Petri net by analysing the projected net. In the remainder of this section we will give two theorems that show how this can be done.

Theorem 3
Let MDPN be a multi-dimensional Petri net and Π_I(MDPN) be an I-projection of MDPN. For any two states $s_1, s_2 \in S$ of MDPN such that s_2 is directly reachable from s_1 (i.e. $s_1 \longrightarrow s_2$), we find that $\Pi_I(s_2)$ is directly reachable from $\Pi_I(s_1)$ (i.e. $\Pi_I(s_1) \longrightarrow \Pi_I(s_2)$) in the I-projection of MDPN.

Proof
Suppose $s_1 \longrightarrow s_2$, then there exists an event $e = \langle t, b_{in}, b_{out}\rangle$ such that $s_1 \xrightarrow{e} s_2$, i.e. $b_{in} \leq s_1$, $\langle b_{in}, b_{out}\rangle \in F_t$ and $s_2 = (s_1 - b_{in}) + b_{out}$.
Let $\overline{e} = \langle t, \Pi_I(b_{in}), \Pi_I(b_{out})\rangle$, then $\Pi_I(s_1) \xrightarrow{\overline{e}} \Pi_I(s_2)$, because $\Pi_I(b_{in}) \leq \Pi_I(s_1)$, $\langle \Pi_I(b_{in}), \Pi_I(b_{out})\rangle \in \overline{F}_t$ and $\Pi_I(s_2) = (\Pi_I(s_1) - \Pi_I(b_{in})) + \Pi_I(b_{out})$.
Therefore, $\Pi_I(s_1) \longrightarrow \Pi_I(s_2)$.
□

Informally speaking, this theorem proves that states directly reachable in a MDPN are also directly reachable in the projected MDPN. Moreover, this theorem implies

that any state reachable from a specific initial state in the MDPN, is also reachable in the projected MDPN. Unfortunately, the opposite does not hold. Not every state in the projected MDPN is also reachable in the MDPN.
Consider for example state s_1 in the MDPN given in section 2.1, with:

$$s_1 \quad = \quad \langle \texttt{waiting}, 'X' \rangle + \langle \texttt{free}, \{'A', 'B'\} \rangle$$

Clearly, this is a 'dead state', i.e. no transition can fire in this state ($'X' \notin \{'A', 'B'\}$). However, in the I_1-projected MDPN defined in this section transition **start** can fire, thus changing state $\prod_I(s_1) = \texttt{waiting} + \texttt{free}$ into state **in_progress** + **busy**.

Nevertheless, we can use theorem 3 to prove that certain states are *not* reachable. For example, let s be a state in a MDPN. If $\prod_I(s)$ is a dead state in $\prod_I$(MDPN), then s is a dead state in MDPN.

It is also possible to find certain invariants by analysing the projected net.

Theorem 4
Let MDPN be a multi-dimensional Petri net and $\prod_I$(MDPN) the I-projection of MDPN. D is the token domain of MDPN and $\overline{D}$ is the token domain of $\prod_I$(MDPN). If $\overline{w} \in \overline{D} \rightarrow \mathbb{Z}$ is an invariant of $\prod_I$(MDPN) and $w \in D \rightarrow \mathbb{Z}$ such that for any $d \in D$: $w(d) = \overline{w}(\pi_I(d))$, then w is an invariant of MDPN.

Proof
Let MDPN $= (D_1, D_2, ..D_n, T, F)$ and $\prod_I(\text{MDPN}) = (D_{i_1}, D_{i_2}, ..D_{i_n}, T, \overline{F})$.
Suppose $\overline{w} \in \overline{D} \rightarrow \mathbb{Z}$ is an invariant of $\prod_I$(MDPN).
Let $w \in D \rightarrow \mathbb{Z}$ such that for any $d \in D$: $w(d) = \overline{w}(\pi_I(d))$. Now we have to prove that for any event $e = \langle t, b_{in}, b_{out} \rangle$ with $\langle b_{in}, b_{out} \rangle \in F_t$:

$$\sum_{d \in D} w(d) b_{in}(d) \quad = \quad \sum_{d \in D} w(d) b_{out}(d)$$

Define $\overline{e} = \langle t, \prod_I(b_{in}), \prod_I(b_{out}) \rangle$. Because $\langle b_{in}, b_{out} \rangle \in F_t$, we infer that $\langle \prod_I(b_{in}), \prod_I(b_{out}) \rangle \in \overline{F}_t$ (see definition 10). Since $\overline{w}$ is an invariant, the following equation holds:

$$\sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})(\textstyle\prod_I(b_{in})(\overline{d})) \quad = \quad \sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})(\textstyle\prod_I(b_{out})(\overline{d})) \qquad (1)$$

Because of the definitions of $\prod_I$ and w, and the fact that D can be partitioned into the sets $\{d \in D \mid \pi_I(d) = \overline{d}\}$ with $\overline{d} \in \overline{D}$, we find that the following mathematical equations are equivalent to equation (1):

$$\sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})((\sum_{d \in D} b_{in}(d)\pi_I(d))(\overline{d})) \quad = \quad \sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})((\sum_{d \in D} b_{out}(d)\pi_I(d))(\overline{d})) \qquad (2)$$

$$\sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})(\sum_{\substack{d \in D \\ \pi_I(d) = \overline{d}}} b_{in}(d)) \quad = \quad \sum_{\overline{d} \in \overline{D}} \overline{w}(\overline{d})(\sum_{\substack{d \in D \\ \pi_I(d) = \overline{d}}} b_{out}(d)) \qquad (3)$$

$$\sum_{d\in D} \overline{w}(\pi_I(d))b_{in}(d) = \sum_{d\in D} \overline{w}(\pi_I(d))b_{out}(d) \quad (4)$$
$$\sum_{d\in D} w(d)b_{in}(d) = \sum_{d\in D} w(d)b_{out}(d) \quad (5)$$

Thus, w is an invariant of MDPN.
□

Consider for example the MDPN given in section 2.1 and the two projected nets $\prod_{I_1}$(MDPN) and $\prod_{I_2}$(MDPN) defined in this section.
By analysing the I_1-projection we find that $w'_1 \in D_{place} \to \mathbb{Z}$ such that $w'_1(\texttt{waiting}) = w'_1(\texttt{in\_progress}) = w'_1(\texttt{ready}) = 1$ and $w'_1(\texttt{free}) = w'_1(\texttt{busy}) = 0$ is an invariant. We can use this to prove that w_1 defined in section 5 is invariant. This invariant shows that the number of tasks is constant.
Consider the I_2-projection of the MDPN. Let $r \in \mathit{Resource}$ be a specific type of resource and let $w \in D_{colour} \to \mathbb{Z}$ be a weight function such that for any $d \in D_{colour}$:

$$w(d) = \begin{cases} 1 & \text{if } d = r \\ 0 & \text{if } d \neq r \end{cases}$$

This weight function is an invariant of the I_2-projection of MDPN. We can use this invariant to prove that there is conservation of resources, i.e. the multi-set of resources in the MDPN does not change.

These examples show that if we use a proper projection we can derive useful properties of a multi-dimensional Petri net. We have shown that it is possible to prove that certain states are not reachable (see theorem 3). Moreover, it is possible to produce invariance properties of a MDPN by analysing some of the projections. These are very important results since analysis of the projected net is often much more easier. This way it may be possible to analyse properties that would have been intractable without a projection.

7 Conclusion

The multi-dimensional Petri net model provides a concept which can be used to deal with extensions like 'colour', 'time', 'identification', etc. in a unifying way. Compared to other Petri net models there are two remarkable differences: (1) the introduction of dimensions and (2) the absence of places. In this paper we have shown how the multi-dimensional Petri net model can be used and how it relates to other net models.
One of the merits of multi-dimensional Petri nets is the ability to define and use projections. By analysing the projected multi-dimensional Petri net we can deduce properties of the original multi-dimensional Petri net. As an example we showed that any invariant of the projected net is also an invariant of the original net.

References

[1] W.M.P. VAN DER AALST, *Timed coloured Petri nets and their application to logistics*, PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.

[2] ——, *Interval Timed Coloured Petri Nets and their Analysis*, in Advances in Petri Nets 1993, M. Ajmone Marsan, ed., vol. 691 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1993, pp. 453–472.

[3] B. BERTHOMIEU AND M. DIAZ, *Modelling and verification of time dependent systems using Time Petri Nets*, IEEE Transactions on Software Engineering, 17 (1991), pp. 259–273.

[4] S. CHEN, J. KE, AND J. CHANG, *Knowledge Reprsentation Using Fuzzy Petri Nets*, IEEE Transactions on Knowledge and Data Engineering, 2 (1990), pp. 311–319.

[5] H.J. GENRICH AND K. LAUTENBACH, *System modelling with high level Petri nets*, Theoretical Computer Science, 13 (1981), pp. 109–136.

[6] K.M. VAN HEE, *System Engineering: a Formal Approach* (to appear), 1993.

[7] K.M. VAN HEE AND P.A.C. VERKOULEN, *Integration of a Data Model and High-Level Petri Nets*, in Proceedings of the 12th International Conference on Applications and Theory of Petri Nets, Aarhus, June 1991, pp. 410–431.

[8] K. JENSEN, *Coloured Petri Nets: A High Level Language for System Design and Analysis*, in Advances in Petri Nets 1990, G. Rozenberg, ed., vol. 483 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1990, pp. 342–416.

[9] ——, *Coloured Petri Nets. Basic concepts, analysis methods and practical use.*, EATCS monographs on Theoretical Computer Science, Springer-Verlag, New York, 1992.

[10] M. AJMONE MARSAN, G. BALBO, AND G. CONTE, *Performance Models of Multiprocessor Systems*, The MIT Press, Cambridge, 1986.

[11] M.K. MOLLOY, *On the Integration of Delay and Throughput Measures in Distributed Processing Models*, PhD thesis, University of California, Los Angeles, 1981.

[12] T. MURATA, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, 77 (1989), pp. 541–580.

[13] J.L. PETERSON, *Petri net theory and the modeling of systems*, Prentice-Hall, Englewood Cliffs, 1981.

[14] C.A. PETRI, *Kommunikation mit Automaten*, PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[15] W. REISIG, *Petri nets: an introduction*, Prentice-Hall, Englewood Cliffs, 1985.

[16] W.M. ZUBEREK, *Timed Petri Nets and Preliminary Performance Evaluation*, in Proceedings of the 7th annual Symposium on Computer Architecture, vol. 8(3) of Quarterly Publication of ACM Special Interest Group on Computer Architecture, 1980, pp. 62–82.

A Appendix

A.1 Multi-sets

A *multi-set*, like a set, is a collection of elements over the same subset of some universe. However, unlike a set, a multi-set allows multiple occurrences of the same element. Another word for multi-set is *bag*. Bag theory is a natural extension of set theory (Jensen [8]).

Definition 11 (multi-sets)
A **multi-set** b, over a set A, is a function from A to $\mathbb{N}$, i.e. $b \in A \rightarrow \mathbb{N}$.[3] If $a \in A$ then $b(a)$ is the number of occurrences of a in the multi-set b. A_{MS} is the set of all multi-sets over A. The empty multi-set is denoted by $\emptyset_A$ (or $\emptyset$). We often represent a multi-set $b \in A_{MS}$ by the formal sum:[4]

$$\sum_{a \in A} b(a)\, a$$

Consider for example the set $A = \{a, b, c, ..\}$, the multi-sets $3a$, $a + b + c + d$, $1a + 2b + 3c + 4d$ and $\emptyset_A$ are members of A_{MS}.

[3] $\mathbb{N} = \{0, 1, 2, ..\}$
[4] This notation has been adopted from Jensen [8].

Definition 12
We now introduce some operations on multi-sets. Most of the set operators can be extended to multi-sets in a rather straightforward way. Suppose A a set, $b_1, b_2 \in A_{MS}$ and $q \in A$:

$q \in b_1$ iff $b_1(q) \geq 1$	(membership)
$b_1 \leq b_2$ iff $\forall_{a \in A}\, b_1(a) \leq b_2(a)$	(inclusion)
$b_1 = b_2$ iff $b_1 \leq b_2$ and $b_2 \leq b_1$	(equality)
$b_1 + b_2 = \sum_{a \in A} (b_1(a) + b_2(a))\, a$	(summation)
$b_1 - b_2 = \sum_{a \in A} ((b_1(a) - b_2(a)) \max 0)\, a$	(subtraction)
$\#b_1 = \sum_{a \in A} b_1(a)$	(cardinality of a finite multi-set)

See Jensen [8, 9] for more details.

A.2 Coloured Petri nets

In section 2.1 we have already seen an example of a coloured Petri net transformed into a multi-dimensional Petri net. In this appendix, we discuss the relation between these two net models in more detail.
We use a slightly modified version of the definition given in Jensen [9]: [5]

Definition 13 (CPN)
A **CP-matrix** is a defined by a tuple CPM $= (P, T, S, C, I_-, I_+)$ satisfying the following requirements:

(i) P is a finite set of places.

(ii) T is a finite set of transitions ($P \cap T = \emptyset$).

(iii) S is a finite set of types, called colour-sets. Each colour-set must have a non-empty and finite set of values.

(iv) C is the colour-function mapping from $P \cup T$ into S. It attaches to each place a set of possible token-colours and to each transition a set of possible occurrence-colours.

(v) I_- and I_+ are the negative and positive incidence-functions defined on $P \times T$, such that $I_-(\langle p,t \rangle), I_+(\langle p,t \rangle) \in C(t) \not\rightarrow C(p)_{MS}$ for all $\langle p,t \rangle \in P \times T$. [6]

The negative incidence-function I_- describes how tokens are removed from places

[5]The domains of the incidence functions $I_-(p,t)$ and $I_+(p,t)$ are not multi-sets, because we do not allow transitions to fire concurrently. That is, we use an interleaving semantics: if two transitions fire concurrently, some non-deterministic ordering is made.

[6]$A \not\rightarrow B$ denotes the set of all partial functions from A to B.

by the occurrence of a transition. The positive incidence-function I_+ is analogous except that it describes how tokens are added to places by the occurrence of a transition. For a more detailed description of coloured Petri nets, the reader is referred to [8, 9].

Now we are able to show how a coloured Petri net is transformed into an equivalent MDPN.

Theorem 5 (CPN $\rightarrow$ MDPN)
Given a CP matrix CPM $= (P, T, S, C, I_-, I_+)$, we can construct an equivalent multi-dimensional Petri net MDPN $= (\overline{D}_{place}, \overline{D}_{colour}, \overline{T}, \overline{F})$ where:

(i) $n = 2$

(ii) $\overline{D}_{place} = P$

(iii) $\overline{D}_{colour} = \bigcup_{p \in P} C(p)$

(iv) $\overline{T} = T$

(v) For all $t \in T$, we define:

$$\begin{aligned} \overline{F}_t &= \{ (\sum_{p \in P} \sum_{c \in C(p)} (I_-(p,t)(b))(c) \langle p, c \rangle, \\ &\quad \sum_{p \in P} \sum_{c \in C(p)} (I_+(p,t)(b))(c) \langle p, c \rangle \ ) \\ &\quad \mid b \in C(t) \ \wedge \ b \in dom(I_-(p,t)) \ \wedge \ b \in dom(I_+(p,t)) \} \end{aligned}$$

The constructed MDPN has two dimensions. Each transition in the coloured Petri net corresponds to precisely one transition in the constructed MDPN. It can be proved that these nets are equivalent.
Naturally, the opposite, i.e. the construction of an equivalent coloured Petri net given a MDPN, is also possible. There are many ways to do this.

A.3 Timed Petri nets

The classical Petri net model is not capable of handling quantitative time. The introduction of coloured Petri nets allowed people to quantify time in an implicit manner, i.e. time is represented by the value or colour of a token. In this case, we have to model a global clock using a place connected to every transition. This place contains one token, whose value represents the current time. Since this is rather cumbersome, many authors have proposed a Petri net model with explicit quantitative time.
We will extend the multi-dimensional Petri net with a time concept by adding a *time* dimension D_{time}. D_{time} is the set of all points of time we want to consider. If $d \in D$ is a token, then $\pi_{time}(d)$ is the **timestamp** of this token. This timestamp

indicates the time a token becomes available. Transitions determine a delay for each produced token. A similar time concepts are used in [1, 2, 6, 8].
To add this time concept we have to replace definition 4 by definition 14:

Definition 14
The **event time** of an event $e = \langle t, b_{in}, b_{out} \rangle \in E$ is the maximum of all time timestamps of the tokens to be consumed:

$$ET(e) \;=\; \max_{d \in b_{in}} \pi_{time}(d)$$

An event $e = \langle t, b_{in}, b_{out} \rangle \in E$ is **enabled** in state $s \in S$ iff:

(i) $b_{in} \leq s$

(ii) $\langle b_{in}, b_{out} \rangle \in F_t$

(iii) For all $d \in b_{in}$ and $d' \in b_{out}$: $\pi_{time}(d) \leq \pi_{time}(d')$.

(iv) All other events which satisfy (i), (ii) and (iii) have an event time of at least $ET(e)$.

The timestamps of the tokens produced by a firing of transition t have to be at least as large as the timestamps of the tokens consumed by transition t (iii). The transition with the smallest enabling time will fire first (iv). If there are multiple transitions having a minimal enabling time, then any of these transitions may be the first to fire. Moreover, transitions are eager to fire, i.e. an event occurs at its event (enabling) time.
In [1] it is shown how this time concept can be used to model all kinds of time-dependent behaviour.

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.

92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.