Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 148 (2006) 151–172

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Transformations of Petri Nets

H. Ehrig, K. Hoffmann, J. Padberg [1]

Institute for Software Technology and Theoretical Computer Science
Technical University Berlin, Germany

Abstract

The aim of this paper is an introduction to the area of Petri net transformations, a rule-based approach for dynamic changes of the net structure of Petri nets. This is especially important for the stepwise construction of Petri nets in the sense of the software development process in software engineering. The concept of Petri net transformations is based on that of graph transformations and high-level replacement systems and it is introduced within a small case study logistics.

Keywords: Petri nets, rule-based approach, transformations, high-level replacement systems, graph transformation

1 Introduction

The main idea of graph transformations is the rule-based modification of graphs where each application of a rule leads to a graph transformation step. Since Petri nets can be considered as bipartite graphs the concept of graph transformations can be applied to define transformations of Petri nets. While the well-known token game of Petri nets does not change the net structure the concept of Petri net transformations is a rule-based approach for dynamic changes of the net structure of Petri nets. Petri net transformations have been introduced in [31] as instantiation of high-level replacement systems [12,13], where the algebraic approach to graph transformations [14] based on double pushouts has been generalised to suitable categories. So, we now start with a general overview of graph transformations, for more see [35,10,15].

[1] Email: {ehrig,hoffmann,padberg}@cs.tu-berlin.de

doi:10.1016/j.entcs.2005.12.016

The research area of graph transformations is a discipline of computer science which dates back to the early seventies. Methods, techniques, and results from the area of graph transformations have already been studied and applied in many fields of computer science such as formal language theory, pattern recognition and generation, compiler construction, software engineering, concurrent and distributed systems modelling, database design and theory, logical and functional programming, AI, visual modelling, etc. Graph transformation has at least three different roots, namely from Chomsky grammars on strings to graph grammars, from term rewriting to graph rewriting, and from textual description to visual modelling.

The main idea is to advocate graph transformations for the whole range of computing. Our concept of *Computing by Graph Transformations* is not limited to programming but includes also specification and implementation by graph transformations as well as graph algorithms and computational models and computer architectures for graph transformations. Computing by graph transformation is a fundamental concept for programming, specification, concurrency, distribution, and visual modelling. A state of the art report for applications, languages and tools for graph transformation on one hand and for concurrency, parallelism and distribution on the other hand is given in volumes 2 and 3 of the *Handbook of Graph Grammars and Computing by Graph Transformation* [10] and [15]. In our paper [17] we have presented a comprehensive prsentation of graph and net transformation and their relation. Petri net transformations can also be realized for algebraic high-level nets [31], which is a high-level net concepts integrating algebraic specifications with place/transition nets.

The main idea of graph transformation as well as of Petri net transformations is the rule-based modification of graphs, respectively of Petri nets. Basically a rule (or production) $p = (L, R)$ is a pair of graphs (or nets) called left hand side L and right hand side R. Applying the rule $p = (L, R)$ means to find a match of L in the source and to replace L by R. In order to replace R by L we need to connect R with the context leading to the target graph (respectively the target net) of the transformation.

In Section 2 we present an intuitive introduction to Petri net transformations by illustrating the rule-based modification of place/transition nets in terms of a small example in the area of logistic processes. The example shows how to use Petri net transformations as vertical refinement concept in the sense of software engineering. Moreover we demonstrate that net transformations are compatible with union of nets. This corresponds to compatibility of vertical and horizontal structuring in the sense of software engineering and enhances the relevance of Petri net transformations for software engineering.

In Section 3 we present precise definitions for the basic notions of Petri net transformations in the case of place/transition nets. The union theorem shows compatibility of net transformations with union of subnets via a common interface provided that the net transformations are preserving this interface. Further results are briefly discussed at the end of Section 3. In the conclusion we discuss some of the future work.

2 Introduction to Petri Net Transformations

In this section we introduce Petri net transformations by example of a case study. In contrast to most applications of the graph transformation approach, where graphs denote states of a system and rules and transformations describe state changes and the dynamic behaviour of systems, in the area of Petri nets we use rules and hence transformations to represent stepwise modification of nets. This kind of transformation for Petri nets is considered to be a vertical structuring technique, known as rule-based net transformation. Rules describe the replacement of a left-hand side net by a right-hand side net. The application of the rule yields a transformation where in the source net the subnet corresponding to the left-hand side is replaced by the subnet corresponding to the right-hand side.

In the following we show how Petri net transformations can be used in the case study logistics before we present the basic concepts in Section 3.

2.1 Case Study Logistics

In this example we illustrate the rule-based modification of place/transition nets in terms of a small example in the area of logistic processes. For the full case study we refer to [6].

Statement of the Case Study Logistics

We consider the logistics of a company consisting of the following departments:

- offer preparation
- order acceptance
- order processing
- shipping department
- accounts receivable

The first version of the logistic process is given by the Petri net in Figure 1. This Petri net is decomposed into five parts corresponding to the departments described above. The union describes the gluing of the subnets along the interface. In this case the interface net consists of places only, so that the union corresponds to the usual place fusion of nets. But the general union construction allows having arbitrary subnets as interfaces.

In the following we modify the subnets independently of each other by applying the rules **r1**-**r4** shown in Fig. 2-6 leading to the new version of each department shown in Fig. 7. To distinguish between previous and new customers the application of rule **r1** adds a place *new customer* and a transition *new customer request*. In contrast rule **r2** expresses at an abstract level that the check of the availability of articles can be done in two different ways in the new version of the logistic process. On one hand the complete order of the customer have to be in the stock, and on the other hand the order is changed is such a way that only those articles are included which are available at the moment.

In the first version of the logistic process depicted in Figure 1 the invoice is created after the shipping document. Because these documents are independent of each other in the new version of the logistic process they should be created in parallel. This is realized by the application of rule **r3**.

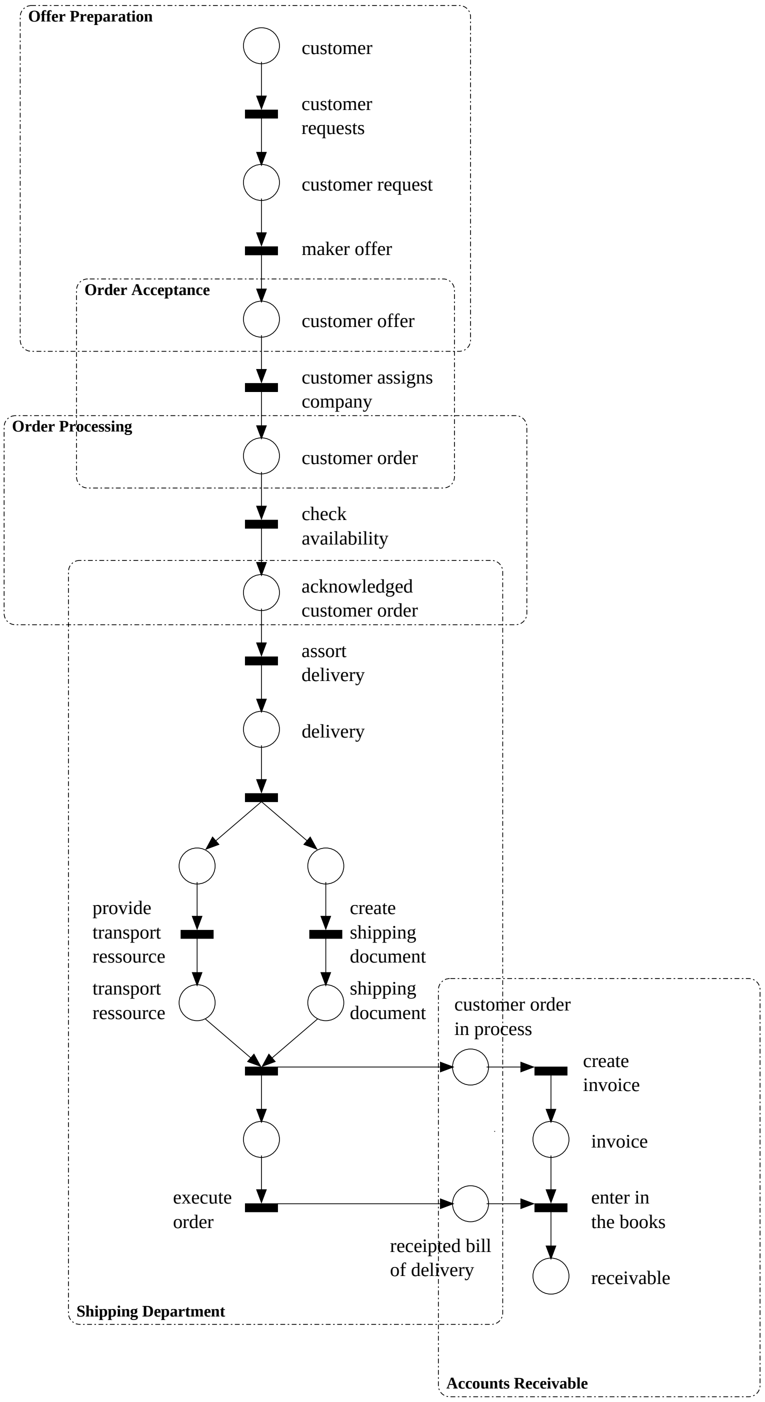

Figure 1. Logistic Process: Net **PN1**

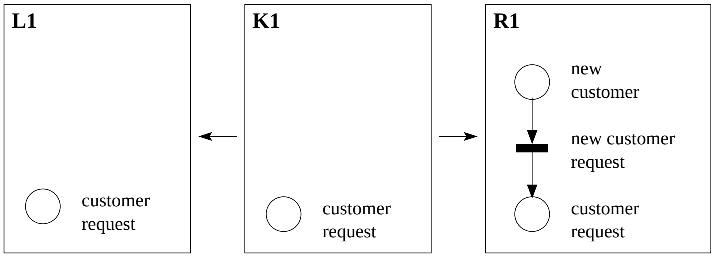

Figure 2. Introducing New Customer: Rule **r1**

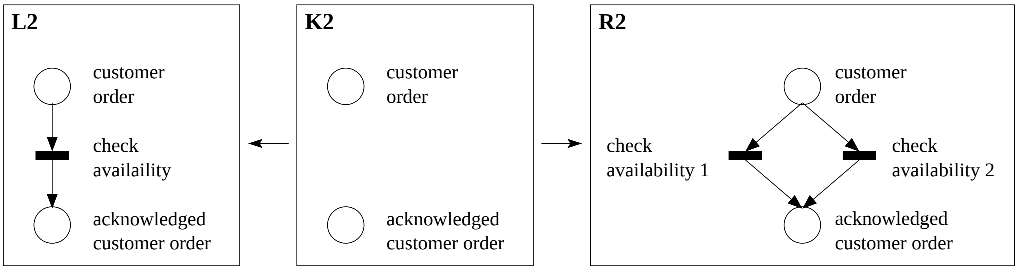

Figure 3. Different Checks of Availability: Rule **r2**

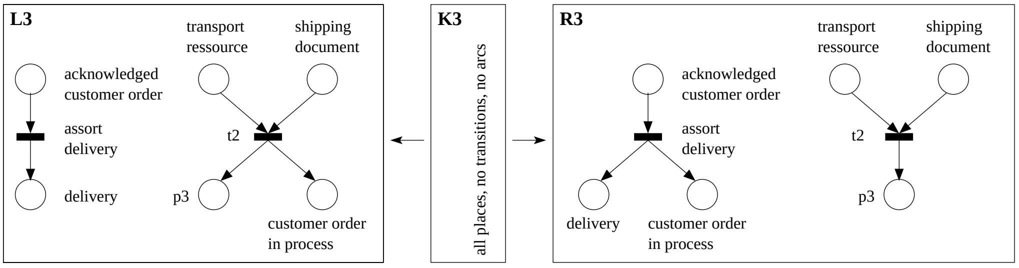
L3
acknowledged customer order
assort delivery
delivery
transport ressource
shipping document
t2
p3
customer order in process
K3
all places, no transitions, no arcs
R3
acknowledged customer order
assort delivery
delivery
customer order in process
transport ressource
shipping document
t2
p3

Figure 4. Create Shipping Documents and Invoice in Parallel: Rule **r3**

We show explicitly the direct transformation with rule **r3** from **Shipping Department 1** (see Fig. 1) to **Shipping Department 2** (see Fig. 7) in Fig. 5. In the upper row of Fig. 5 we show rule **r3**. In a first step we delete from **Shipping Department 1** both transitions of rule **r3** and adjacent edges, but we preserve all places of **L3**, because they are also in **K3** and **R3**, leading to the context net **C** in Fig. 5. In a second step we glue together **C** and **R3** via

K3 leading to **Shipping Department 2** in Fig. 5.

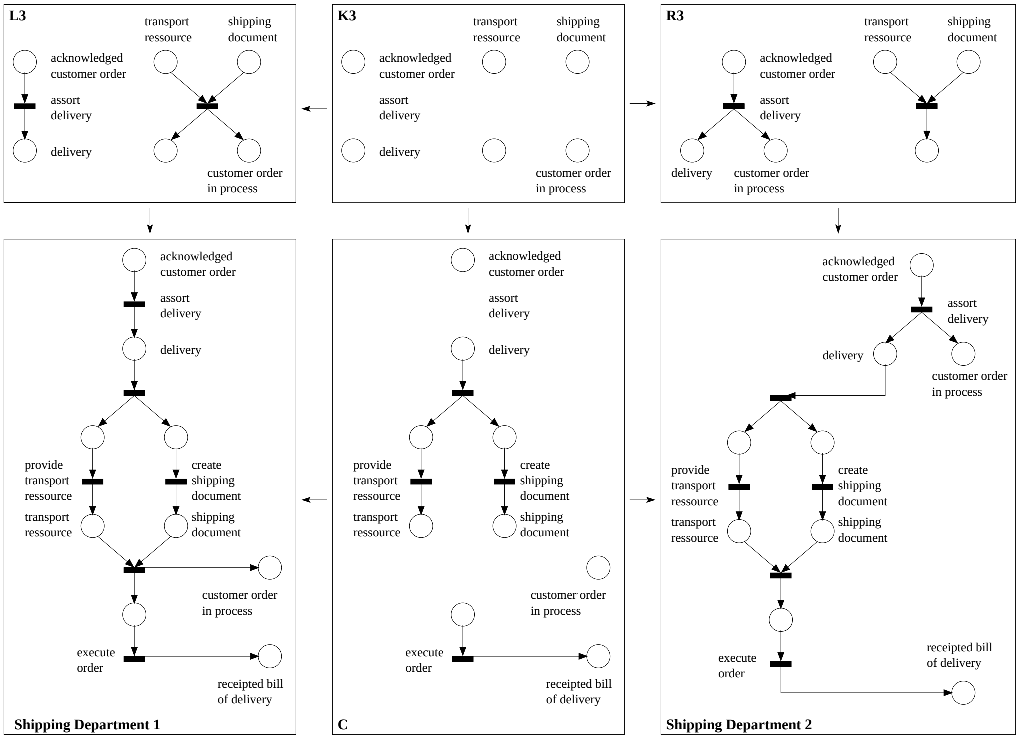

Figure 5. Direct Transformation: **Shipping Dep. 1** $\overset{r3}{\Longrightarrow}$ **Shipping Dep. 2**

Finally, the area of responsibility of the department **Accounts Receivable** is expanded to check the correct payments of customers by rule **r4** in Fig. 6.

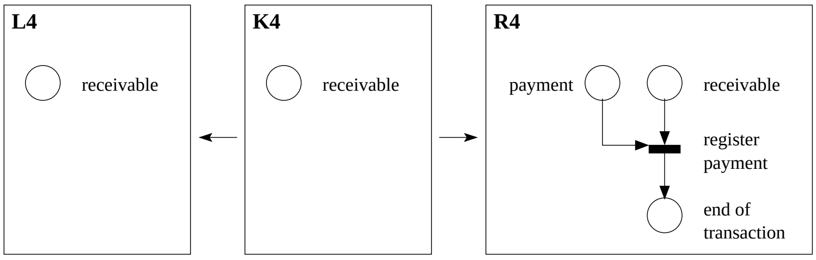

Figure 6. Introducing Payment: Rule **r4**

Since all rules and transformations are preserving the interfaces of the corresponding union in Fig. 1, esp. rule **r1** preserves the place *customer offer*, the interfaces are still available in Fig. 7 and can be used to construct the resulting net **PN2**. The union theorem in Section 3 makes sure that this construction leads to the same result as if we would have applied the rules **r1**-**r4** sequentially to the entire net **PN1** in Fig. 1. This is a typical example for compatibility of horizontal structuring (union) with vertical refinement (rule-based transformation).

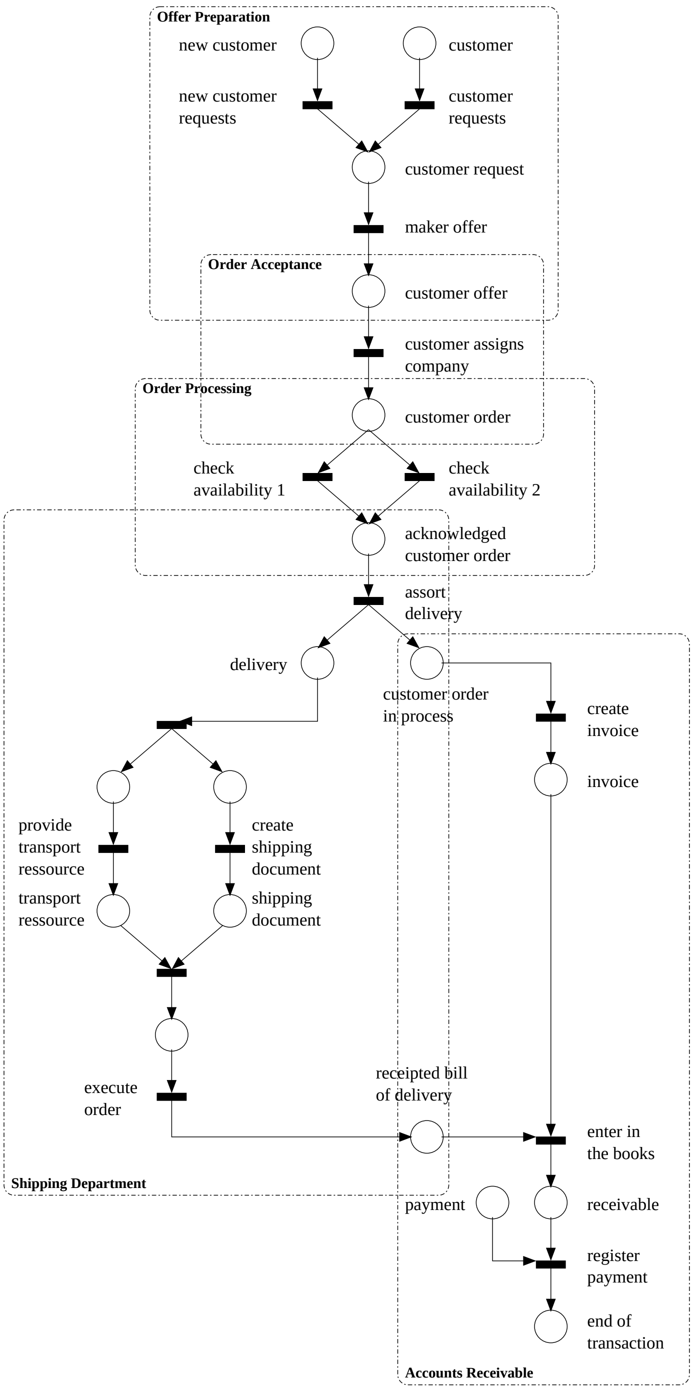

Figure 7. Resulting Net **PN2**

2.2 Relevance of Petri Net Transformations

The above example illustrates only some of the possibilities and advantages of net transformations. The usual argument in favour of formal techniques, to have precise notions and valid results clearly holds for this approach as well. Moreover, we have already investigated net transformations in high-level Petri net classes (see Section 3.6) that are even more suitable for system modelling than the place/transition nets in our example. The impact for system development is founded in what results from net transformations:

- *Stepwise Development of Models:* The model of a complex software system may reach a size that is difficult to handle and may compromise the advantages of the (formal) model severely. The one main counter measure is breaking down the model into sub-models, the other is to develop the model top-down. In top-down development the first model is a very abstract view of the system and step by step more modelling details and functionality are added. In general however, this results in a chain of models, that are strongly related by their intuitive meaning, but not on a formal basis. Petri net transformations fill this gap by supporting the step-by-step development of a model formally. Rules describe the required changes of a model and their application yields the transformations of the model. Moreover, the representation of change in a visual way using rules and transformations is very intuitive and does not require a deeper knowledge of the theory.
- *Distributed Development of Models:* Decomposing a model, that is too large, is an important technique for the development of complex models. To combine the advantages of a horizontal structuring with the advantages of step-by-step development techniques for ensuring the consistency of the composed model are required. Then a distributed step-by-step development is available, that allows the independent development of sub-models. The theory of net transformations comprises horizontal structuring techniques and ensures compatibility between these and the transformations. In our example we have employed the union construction for the decomposition, and have subsequently developed the subnets independently of each other. The theory allows much more complex decompositions, where the independence of the sub-models is not as obvious as in the given example. So, the formal foundation for the distributed development of complex models is given.
- *Incremental Verification:* Pure modification of Petri nets is often not sufficient, since the net has some desired properties that have to be ensured

during further development. Verification of each intermediate model requires a lot of effort and hence is cost intensive. But refinement can be considered as the modification of nets preserving desired properties. Hence the verification of properties is only required for the net, where they can be first expressed. In this way properties are introduced into the development process and are preserved from then on. Rule-based refinement modifies Petri nets using rules and transformations so that specific system properties are preserved. For a brief discussion see Section 3.6.

- *Foundation for Tool Support:* A further advantage is the formal foundation of rule-based refinement and/or rule-based modification for the implementation of tool support. Due to the theory of Petri net transformations we have a precise description, how rules and transformation work on Petri nets. Tool support is for the practical use the main precondition. The user should get tool support for defining and applying rules. The tool should assist the choice as well as the execution of rules and transformations.
- *Variations of the Development Process:* Another area, where transformations are very useful, concerns variations in the development process. Often a development is not entirely unique, but variations of the same development process lead to variations in the desired models and resulting systems. These variations can be expressed by different rules yielding different transformations, that are used during the step-by-step development.

3 Concepts of Petri Net Transformations

In this section we give the precise definitions of the notions that we have already used in our example. For notions and results beyond that we give a brief survey in Section 3.6 and refer to literature.

The concept of Petri net transformations is a special case of high-level replacement systems. High-level replacement systems have been introduced in [12] as a categorical generalisation of the double-pushout approach to graph transformation. The theory of high-level replacement systems can successfully be employed not only to graph transformation, but also to other areas, as Petri nets (see [12]). This leads to the concept of Petri net transformations as instantiation of high-level replacements systems. In the following we present explicitly the resulting concept of Petri net transformations.

3.1 Place/Transition Nets and Net Morphisms

Let us first present a notation of place/transition net that is suitable for our transformation approach. We assume that the nets are given in the algebraic style as introduced in [27]. A place/transition net $N = (P, T, pre, post)$ is given by the set of places P, the set of transitions T, and two mappings $pre, post : T \to P^{\oplus}$, the pre-domain and the post-domain,

$$\begin{tikzcd} T \arrow[r, shift left, "pre"] \arrow[r, shift right, "post"'] & P^{\oplus} \end{tikzcd},$$

where $P^{\oplus}$ is the free commutative monoid over P that can also be considered as the set of finite multisets over P. The pre- (and post-) domain function maps each transition into the free commutative monoid over the set of places, representing the places and the arc weight of the arcs in the pre-domain (respectively in the post-domain). For finite P an element $w \in P^{\oplus}$ can be presented as a linear sum $w = \sum_{p \in P} \lambda_p \cdot p$ with $\lambda_p \in \mathbb{N}$ or as a function $w : P \to \mathbb{N}$. In the infinite case we have to require that $\lambda_p \neq 0$ only for finitely many $p \in P$ that means the corresponding $w : P \to \mathbb{N}$ has finite support.

Based on the algebraic notion of Petri nets we use simple homomorphisms that are generated over the set of places. These morphisms map places to places and transitions to transitions. The pre-domain of a transition has to be preserved, that is even if places may be identified the number of tokens that are taken, remains the same. This is expressed by the condition $pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$.

A morphism $f : N_1 \to N_2$ between two place/transition nets $N_1 = (P_1, T_2, pre_1, post_1)$ and $N_2 = (P_2, T_2, pre_2, post_2)$ is given by $f = (f_P, f_T)$ with $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ so that $pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$ and $post_2 \circ f_T = f_P^{\oplus} \circ post_1$. The diagram schema for net morphisms is given in the following diagram.

$$\begin{tikzcd} T_1 \arrow[r, shift left, "pre_1"] \arrow[r, shift right, "post_1"'] \arrow[d, "f_T"] & P_1^{\oplus} \arrow[d, "f_P^{\oplus}"] \\ T_2 \arrow[r, shift left, "pre_2"] \arrow[r, shift right, "post_2"'] & P_2^{\oplus} \end{tikzcd}$$

Several examples of net morphisms can be found in Figure 5 where the horizontal and vertical arrows denote injective net morphisms.

3.2 Rules and Transformations

The category **PT** consists of place/transition nets as objects and place/transition net morphisms as morphisms. In order formalise rules and transformations for nets in the DPO-approach we first state the construction of pushouts in the category **PT** of place/transition nets. For any span of two morphisms $N_1 \leftarrow N_0 \rightarrow N_2$ the pushout can be constructed. The construction is based on the pushouts for the sets of transitions and sets of places in the category **Set** of sets. In the category **Set** sets and functions the pushout object D is given by the quotient set

$D = B + C/\equiv$, short $D = B +_A C$,

where $B + C$ is the disjoint union of B and C and $\equiv$ the equivalence relation generated by $f(a) \equiv g(a)$ for all $a \in A$. In fact D can be interpreted as the gluing of B and C along A: Starting with the disjoint union $B + C$ we glue together the elements $f(a) \in B$ and $g(a) \in C$ for each $a \in A$.

Given the morphisms $f : N_0 \to N_1$ and $g : N_3 \to N_2$ then the pushout N_3 in the category **PT** with the morphisms $f' : N_2 \to N_3$ and $g' : N_1 \to N_3$ is constructed (see digram below) as follows:

- $T_3 = T_1 +_{T_0} T_2$ with f'_T and g'_T as pushout of f_T and g_T in **Set**.
- $P_3 = P_1 +_{P_0} P_2$ with f'_P and g'_P as pushout of f_P and g_P in **Set** as well.
- $pre_3(t) = \begin{cases} [pre_1(t_1)] & ; \text{ if } g'_T(t_1) = t \\ [pre_2(t_2)] & ; \text{ if } f'_T(t_2) = t \end{cases}$
- $post_3(t) = \begin{cases} [post_1(t_1)] & ; \text{ if } g'_T(t_1) = t \\ [post_2(t_2)] & ; \text{ if } f'_T(t_2) = t \end{cases}$

$$\begin{tikzcd}
N_0 \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(=)"] & N_1 \arrow[d, "g'"] \\
N_2 \arrow[r, "f'"'] & N_3
\end{tikzcd}$$

Two examples of the pushout construction of nets are depicted in Figure 5. We have the embedding of **K3** into **L3** and **C**. The pushout describes the gluing of the nets **L3** and **C** along the six places of the interface **K3**. Hence we have the pushout $\mathbf{L3} +_{\mathbf{K3}} \mathbf{C}$ =**Shipping Department 1** on the left hand side of 5. Similarly we have the pushout $\mathbf{R3} +_{\mathbf{K3}} \mathbf{C}$ =**Shipping Department 2** on the right hand side of Figure 5.

Next we introduce rules, that correspond to graph productions in the DPO-approach. Rules describe the replacement of the left-hand side net by the right-hand side net in the presence of an interface net.

- A rule $r = (L \xleftarrow{k_1} K \xrightarrow{k_2} R)$ consists of place/transition nets L, K and R, called left-hand side, interface and right-hand side net respectively, and two injective net morphisms $K \xrightarrow{k_1} L$ and $K \xrightarrow{k_2} R$.
- Given a rule $r = (L \xleftarrow{k_1} K \xrightarrow{k_2} R)$ a direct transformation $N_1 \overset{r}{\Longrightarrow} N_2$, from N_1 to N_2 is given by two pushout diagrams (1) and (2) in the following diagram. The morphisms $m : L \to N_1$ and $n : R \to N_2$ are called match and comatch, respectively. The net C is called pushout complement or the

context net.

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(\mathbf{1})"] & K \arrow[l, "k_1"'] \arrow[r, "k_2"] \arrow[d, "c"] \arrow[dr, phantom, "(\mathbf{2})"] & R \arrow[d, "n"] \\
N_1 & C \arrow[l] \arrow[r] & N_2
\end{tikzcd}$$

The illustration of a transformation can be found for our example in Figure 5, where the rule **r3** is applied to the net **Shipping Department 1** with match m. As explained above the first pushout denotes the gluing of the nets **L3** and **C** along the net **K3** resulting in the net **Shipping Department 1**. The second pushout denotes the gluing of the nets **R3** and **C** along the net **K3** resulting in the net **Shipping Department 2**.

3.3 Gluing Condition and the Construction of the Context Net

Given a rule r and a match m as depicted in the diagram above, then we construct in a first step the pushout complement provided the gluing condition holds. This leads to the pushout (1) in the diagram above. In a second step we construct the pushout of c and k_2 leading to N_2 and the pushout (2) in the diagram above.

The gluing condition correspond exactly to the gluing condition in the graph case. Using the same interpretation as in the graph case, but the notation in Subsection 3.2 we have the following:

Gluing Condition for Nets:

$$BOUNDARY \subseteq GLUING$$

where $BOUNDARY$ and $GLUING$ are subnets of L defined by

- $GLUING = k_1(K)$
- $DANGLING = \{p \in P_L \mid \exists t \in T_1 - m_T(T_L) :$
  $(m_P(p) \in pre_1(t) \text{ or } m_P(p) \in post_1(t))\}$
  where the notation $p \in pre_1(t)$ means $pre_1(t) = \sum_{p \in P_1} \lambda_p \cdot p$ with $\lambda_p > 0$, similar for $post_1$,
- $IDENTIFICATION = \{x \in K \mid \exists y \in K : (x \neq y \text{ and } m(x) = m(y))\}$, where $x \in K$ means $x \in P_K$ with $m = m_P$ or $x \in T_K$ with $m = m_T$, and
- $BOUNDARY = DANGLING \cup IDENTIFICATION$

Now the pushout complement C is constructed by:

- $P_C = (P_1 \setminus m_P(P_L)) \cup m_P(k_{1P}(P_K))$
- $T_C = (T_1 \setminus m_T(T_L)) \cup m_T(k_{1T}(T_K))$

- $pre_C = pre_{1|T_C}$ and $post_C = post_{1|T_C}$

Note that the pushout complement C leads to the pushout (1) in the diagram above and that it is unique up to isomorphism.

In our example of the development of the logistic process in Section 2 the gluing condition is satisfied in all cases, since the matches are all injective and places are not deleted by our rules. In fact $DANGLING$ of the match in Fig. 5 is given by all places of **L3** except *acknowledged customer order* and *customer order in process*, while $GLUING$ consists of all places in **L3**. $IDENTIFICATION$ is empty, because the match is injective hence we have $BOUNDARY = DANGLING \subseteq GLUING$.

3.4 Union Construction

The union of two Petri nets sharing a common subnet, that may be empty, is defined by the pushout construction for nets. The union of place/transition nets N_1, N_2 sharing an interface net I with the net morphisms $f : I \to N_1$ and $g : I \to N_2$ is given by the pushout diagram (1) below. Subsequently we use the short notation $N = N_1 +_I N_2$ or $N_1, N_2 \overset{I}{\rightarrowtail\!\!\Longrightarrow} N$.

$$\begin{tikzcd}
I \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & N_1 \arrow[d, "g'"] \\
N_2 \arrow[r, "f'"'] & N
\end{tikzcd}$$

In our example in Fig. 1 we use the union construction to describe the composition of subnets. The interface net I between the subnets **Offer Preparation** and **Order Acceptance** is given by the net consisting only of the place *customer offer*. The other interfaces in Fig. 1 are given by the places *costumer order*, *acknowledged customer order* and the two places *customer order in process* and *receipted bill of delivery*.

3.5 Union Theorem

The Union Theorem states the compatibility of union and net transformations: Given a union $N_1 +_I N_2 = N$ and net transformations $N_1 \overset{r_1}{\Longrightarrow} M_1$ and $N_2 \overset{r_2}{\Longrightarrow} M_2$ then we have a parallel rule $r_1 + r_2 = (L_1 + L_2 \leftarrow K_1 + K_2 \to R_1 + R_2)$, where $N_1 + N_2$ is the disjoint union of nets, and a parallel net transformation $N \overset{r_1+r_2}{\Longrightarrow} M$. Then $M = M_1 +_I M_2$ is the union of M_1 and M_2 with the shared interface I, provided that the given net transformations preserve the interface I. The Union Theorem is illustrated in the following diagram and especially stated and proven in [28]:

$$\begin{tikzcd}
N_1, N_2 \arrow[r, Rightarrow, tail, "I"] \arrow[d, Rightarrow, "{r_1,r_2}"'] \arrow[dr, phantom, "(=)"] & N \arrow[d, Rightarrow, "r_1+r_2"] \\
M_1, M_2 \arrow[r, Rightarrow, tail, "I"] & M
\end{tikzcd}$$

Note that the compatibility requires an independence condition stating that nothing from the interface net I may be deleted by one of the transformations of the subnets. This is obviously the case in our example in Section 2, since the interfaces consist of places only and the rules preserve all places. In fact, we have to apply the union theorem four times in order to obtain the transformation from net **PN1** in Fig. 1 to net **PN2** in Fig. 7.

3.6 Further Results

We briefly introduce the main net classes which have been studied up to now, and subsequently we present some main results.

- Place/transition nets in the algebraic style have already been introduced in Subsection 3.1.
- Coloured Petri nets [23,24,25] are widely known and very popular. Their practical relevance is very high, due to the very successful tool Design/CPN [22].
- Algebraic high-level nets are available in quite a few different notions e.g. [37,34,31]. We use a notion that reflects the paradigm of abstract data types into signature and algebra. An algebraic high-level net (as in [31]) is given by $N = (SPEC, P, T, pre, post, cond, A)$, where $SPEC = (S, OP, E; X)$ is an algebraic specification in the sense of [16] with additional variables X not occurring in E, P is the set of places, T is the set of transitions, $pre, post : T \to (T_{OP}(X) \times P)^{\oplus}$ are the pre- and post-domain mappings, $cond : T \to \mathcal{P}_{fin}(EQNS(SIG, X))$ are the transition guards, and A is a $SPEC$ algebra.

Horizontal Structuring

Union and fusion are two categorical structuring constructions for place/transition nets, that merge two subnets or two different nets into one.

The union has been introduced in the previous subsection. Now let us consider fusion: Given a net F that occurs in two copies in the net N_1, represented by two morphisms $F \underset{f'}{\overset{f}{\rightrightarrows}} N_1$ the fusion construction leads to a net N_2, where both occurrences of F in N_1 are merged. If F consists of places $p_1, .., p_n$ then each of the places occurs twice in net N_1, namely as $f(p_1), ..., f(p_n)$ and $f'(p_1), ..., f'(p_n)$. N_2 is obtained from net N_1 fusing both occurrences $f(p_i)$ and $f'(p_i)$ of each place p_i for $1 \leq i \leq n$.

The Union Theorem is presented in the previous section. The Fusion Theorem [29] is expressed similarly: Given a rule r and a fusion $F \rightrightarrows N_1$ then we obtain the same result whether we derive first $N_1 \overset{r}{\Longrightarrow} N_1'$ and then construct the fusion $F \rightrightarrows N_1'$ resulting in N_2' or whether we construct the fusion $F \rightrightarrows N_1$ first, resulting in N_2 and then perform the transformation step $N_2 \overset{r}{\Longrightarrow} N_2'$. Similar to the Union Theorem a certain independence condition is required. Both theorems state that Petri nets transformations are compatible with the corresponding structuring technique under suitable independence conditions. Roughly spoken these conditions guarantee that the interface net I and respectively the fusion net F are preserved by all net transformations.

Parallelism

We are able to model interleaving and parallelism of net transformations.

The Local Church-Rosser Theorem states a local confluence in the sense of formal languages corresponding to interleaving. The required condition of parallel independence means that the matches of both rules overlap only in parts that are not deleted. Sequential independence means that those parts created by the first transformation step are not deleted in the second. The Parallelism Theorem states that sequential or parallel independent transformations can be carried out either in arbitrary sequential order or in parallel. In the context of step-by-step development these theorems are important as they provide conditions for the independent development of different parts or views of the system. More details for horizontal structuring or parallelism are given in [31] and [29].

Refinement

The extension of high-level replacement systems to rules and transformations preserving properties has the following impact on Petri nets: Rule-based refinement comprises the transformation of Petri nets using rules while preserving certain net properties. For Petri nets the desired properties of the net model can be expressed, e.g in terms of Petri nets (as liveness, boundedness etc.), in terms of logic (e.g. temporal logic, logic of actions etc.) in terms of relation to other models (e.g. bisimulation, correctness etc.) and so on.

For place/transition nets, algebraic-high level nets and Coloured Petri nets the following results for rule-based refinement are presented in the following table. For more details see [33].

Notion/Results	PT-nets	AHL-nets	CPNs
Rules, Transformations	$\surd$	$\surd$	$\surd$
Safety property preserving transformations with			
transition-gluing morphisms	$\surd$	$\surd$	$\surd$
place-preserving morphisms	$\surd$	$\surd$	$\surd$
Safety property introducing transformations	$\surd$	$\surd$	$\surd$
Liveness preserving transformations	$\surd$	?	?
Liveness introducing transformations	$\surd$	?	?
Local Church Rosser I + II Theorem	$\surd$	$\surd$	$\surd$
Parallelism Theorem	$\surd$	$\surd$	$\surd$
Union	$\surd$	$\surd$	$\surd$
Fusion	$\surd$	$\surd$	$\surd$
Union Theorem	$\surd$	$\surd$	$\surd$
Fusion Theorem	$\surd$	$\surd$	$\surd$

Table 1
Achieved results

4 Conclusion

The main idea of Petri net transformations is to extend the classical theory of Petri nets by a rule-based technique that allows modelling the changes of the Petri net structure.

There have been already a few approaches to describe transformations of Petri nets formally (e.g. in [4,5,36,7,38]). The intention has been mainly on reduction of nets to support verification, and not on the software development process as in our case. This use of transformations has been one of the main focus areas of the DFG-Research group *Petri Net Technology*. There are some large studies in various application areas as medical information systems [18], as train control systems [32] or as sketched in this paper in logistics. These case study clearly show the advantages using net transformation in system development and the practical use of the results stated in table 1.

Although the area of Petri net transformations is already well-established, there are many promising directions for further research to follow, for example:

- Transfer to other net classes
  There is a large variety of Petri net classes, and in principle the idea of Petri net transformation is applicable to all. The concept of transformation we have employed is an algebraic one, so the use of algebraic approaches to Petri nets is more suggesting. Algebraic higher-order Nets [21] have been recently developed and are one of the promising targets to transfer the idea of transformations to. These nets extend algebraic high-level nets as they are equipped with a higher-order signature and algebra. This allows most interesting applications and supports structure flexibility and system adaptibilty in an extensive way.
- Component technology
  Components present an advanced paradigm for the structuring of complex systems and have been advocated in the recent years most strongly. Components, that use Petri nets for the specification of the interfaces and the component body have been defined in [30]. There are three nets that represent the import, the export and the body of the component. The export is an abstraction of the body and the import is embedded into the body. There are two operations; the hierarchical composition and the union of components. Unfortunately there is no transformation concept in the sense of graph and net transformation up to now. Based on net transformations the transformation of the import, the export and the body can be defined straightforward.
- Tool support
  The practical use of graph transformations is supported by several tools. The algebraic approach to graph transformations is especially supported by the graph transformation environment AGG (see the homepage of [1]). On top the graph transformation system AGG there is the GenGED environment (see the homepage of [20]) that supports the generic description of visual modelling languages for the generation of graphical editors and the simulation of the behaviour of visual models. Especially, Petri net transformations can be expressed using GenGED, e.g. for the animation of Petri nets [9,3]. In this framework, the animation view of a system modeled as a Petri net consists of a domain-specific layout and an animation according to the firing behaviour of the Petri net. This animation view can be coupled to other Petri net tools [8] using the Petri Net Kernel [26] a tool infrastructure

for editing, simulating and analysing Petri nets of different net classes and for integration of other Petri net tools.

References

[1] AGG Homepage. http://tfs.cs.tu-berlin.de/agg.

[2] P. Baldan, A. Corradini, U. Montanari, F. Rossi, H. Ehrig, and M. Löwe. Concurrent Semantics of Algebraic Graph Transformations. In G. Rozenberg, editor, *The Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[3] R. Bardohl and C. Ermel. Scenario Animation for Visual Behavior Models: A Generic Approach Applied to Petri Nets. In G. Juhas and J. Desel, editors, *Proc. 10th Workshop on Algorithms and Tools for Petri Nets (AWPN'03)*, 2003.

[4] G. Berthelot. Checking properties of nets using transformations. *Advances in Petri Nets 1985*, Lecture Notes in Computer Science 222: pages 19–40. Springer 1986.

[5] G. Berthelot. Transformations and decompositions of nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets*, Lecture Notes in Computer Science 254, pages 359–376. Springer, 1987.

[6] J. Bogen. *Schrittweise Entwicklung von Ereignisgesteuerten Prozessketten zu Algebraischen Higher-Order Netzen*. Master Thesis, Technische Universität Berlin, 2004.

[7] R. David and H. Alla, editors. *Petri Nets and Grafcet*. Master Thesis, Technische Universität Berlin, 2004.

[8] C. Ermel, R. Bardohl, and H. Ehrig. Specification and implementation of animation views for Petri nets. In DFG Research Group *Petri Net Technology, Proc. of 2nd International Colloquium on Petri Net Technology for Communication Based Systems*, 2001.

[9] C. Ermel, R. Bardohl, and H. Ehrig. Generation of animation views for Petri nets in GenGED. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Advances in Petri Nets: Petri Net Technologies for Modeling Communication Based Systems*, Lecture Notes in Computer Science 2472. Springer, 2003.

[10] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

[11] H. Ehrig, M. Gajewsky, and F. Parisi-Presicce. *High-level replacement systems with applications to algebraic specifications and Petri nets*, In [2], chapter 6, pages 341–400.

[12] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comp. Science*, 1:361–404, 1991.

[13] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comp. Science*, 1:361–404, 1991.

[14] H. Ehrig. Introduction to the algebraic theory of graph grammars (A survey). In *Graph Grammars and their Application to Computer Science and Biology*, pages 1–69. Lecture Notes in Computer Science 73. Springer, 1979.

[15] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution*. World Scientific, 1999.

[16] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1985.

[17] H. Ehrig and J. Padberg. *Graph Grammars and Petri Net Transformations.*, In Lectures on Concurrency and Petri Nets Special Issue Advanced Course PNT, pages 496-536. Lecture Notes in Computer Science 73. Springer, 2004.

[18] C. Ermel, J. Padberg, and H. Ehrig. Requirements Engineering of a Medical Information System Using Rule-Based Refinement of Petri Nets. In D. Cooke, B.J. Krämer, P. C-Y. Sheu, J.P. Tsai, and R. Mittermeir, editors, *Proc. Integrated Design and Process Technology*, pages 186–193. Society for Design and Process Science, 1996. Vo

[19] H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973. l.1.

[20] GenGED Homepage. http://tfs.cs.tu-berlin.de/genged.

[21] Kathrin Hoffmann. *Formal approach and applictions of algebraic higher-order nets.* PhD thesis, Technische Universität Berlin, 2005. submitted.

[22] K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN. A Reference Manual.* Meta Software Cooperation, 125 Cambridge Park Drive, Cambridge Ma 02140, USA, 1991.

[23] K. Jensen. *Coloured Petri nets. Basic Concepts, Analysis Methods and Practical Use*, Volume 1: Basic Concepts. Springer Verlag, EATCS Monographs in Theoretical Computer Science edition, 1992.

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

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

[26] E. Kindler and M. Weber. The Petri net kernel – an infrastructure for building Petri net tools. *Software Tools for Technology Transfer*, 3(4):486–497, 2001.

[27] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88(2):105–155, 1990.

[28] J. Padberg. Abstract Petri Nets: Uniform Approach and Rule-Based Refinement. PhD Thesis, Technische Universität Berlin, Germany, 1996.

[29] J. Padberg. Categorical approach to horizontal structuring and refinement of high-level replacement systems. *Applied Categorical Structures*, 7(4):371–403, December 1999.

[30] J. Padberg. Petri net modules. *Special Issue on Component Based System Development, Journal on Integrated Design and Process Technology*, 2002.

[31] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic high-level net transformation systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[32] J. Padberg, P. Schiller, and H. Ehrig. New Concepts for High-Level Petri Nets in the Application Domain of Train Control. In E. Schnieder and U. Becker, editors, *Proc. Vol. 2, 9th Symposium on Transportation Systems*, pages 153–160, 2000.

[33] J. Padberg and M. Urbášek. Rule-based refinement of Petri nets: A survey. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Advances in Petri Nets: Petri Net Technologies for Modeling Communication Based Systems*, Lecture Notes in Computer Science 2472. Springer, 2003.

[34] W. Reisig. Petri Nets and Algebraic Specifications. *Theoretical Computer Science*, 80:1–34, 1991.

[35] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[36] Vanio M. Savi and Xiaolan Xie. Liveness and boundedness analysis for petri nets with event graph modules. In Jensen, K., editor, *13th International Conference on Application and Theory of Petri Nets 1992, Sheffield, UK*, Lecture Notes in Computer Science 616, pages 328–347. Springer, 1992.

[37] J. Vautherin. Parallel system specification with coloured Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 87*, pages 293–308. Lecture Notes in Computer Science 266. Springer, 1987.

[38] W.M.P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 1248, pages 407–426. Springer, 1997.