Electronic Communications of the EASST
Volume 67 (2014)

Proceedings of the
13th International Workshop on Graph Transformation
and Visual Modeling Techniques
(GTVMT 2014)

Lattice-extended Coloured Petri Net Rewriting for Adaptable User Interface Models

Jan Stückrath, Benjamin Weyers

13 pages

Guest Editors: Frank Hermann, Stefan Sauer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Lattice-extended Coloured Petri Net Rewriting for Adaptable User Interface Models

Jan Stückrath[1], **Benjamin Weyers**[2]

[1]jan.stueckrath@uni-due.de
Abteilung INKO, Universität Duisburg-Essen, Duisburg, Germany

[2]weyers@vr.rwth-aachen.de
Virtual Reality Group, RWTH Aachen University, Aachen, Germany

Abstract: Adaptable user interfaces (UI) have shown a great variety of advantages in human computer interaction compared to classic UI designs. We show how adaptable UIs can be built by introducing coloured Petri nets to connect the UI's physical representation with the system to be controlled. UI development benefits from formal modelling approaches regarding the derived close integration of creation, execution, and reconfiguration of formal UI models. Thus, adaptation does not only change the physical representation, but also the connecting Petri net. For the latter transformation, we enhance the DPO rewriting formalism by using an order on the set of labels and softening the label-preserving property of morphisms, i.e., an element can also be mapped to another element if the label is larger. We use lattices to ensure correctness and state application conditions of rewriting steps. Finally we define an order compatible with our framework for the use in our implementation.

Keywords: Coloured Petri Nets, Rewriting, User Interface Modelling, Redesign and Reconfiguration, Lattices

1 Introduction

Modelling in user interface creation has a long tradition, starting in the 1980th with cognitive architectures, such as GOMS and CTT [CMN80, KP85], or later approaches for modelling user-system dialogues [JWZ93]. As known from software engineering [HJSW10], the gap between model and implementation is often a great issue in the design of systems, also affecting the creation of interactive systems and user interfaces. One possible solution to reduce this gap is the use of formal modelling approaches, which can be executed on the computer without further need of extra implementation. Various examples can be found regarding creation and modelling of user interfaces using formal methods, such as works published by Navarre et al. [NPLB09]. Still, resulting models are often inflexible and static, lacking of formal and model-intrinsic adaptation and reconfiguration approaches. Especially regarding the implementation of adaptable UIs, a full-fledged modelling and reconfiguration concept is necessary. Adaptable UIs extend UIs by software support to enable the user to change the UI according to his preferences.

For this we developed a coloured Petri net-based [Jen97] modelling approach for creating formal user interface models [Wey12, WBLK12] accompanied with an extended graph rewriting concept introduced in this paper. The combination of using coloured Petri nets and rewriting creates an executable modelling approach paired with a formal adaptation allowing the creation of flexible and reconfigurable user interface models, thus realising adaptable UIs.

Rewriting of P/T nets has already been considered for instance in [EHP06, LO04], where the latter focuses on linking transformations of the nets structure and marking. Properties of coloured Petri nets, such as types or guard conditions, are modelled by labels on places, transitions and arcs, and a greater modelling flexibility can be achieved by a formalism including the possibility of relabelling. However, this is in conflict with the often used restriction that rules (i.e. morphisms) preserve labels. One approach is to allow non-labelled elements in rules, which is done in [HP02, Ros75], such that the label of an element will change if its interface node is unlabelled. In our approach we introduce an order on the labels (later called inscriptions) and allow rules to be applied to elements with possibly larger labels. With sufficiently complex labels, a rewriting step can partly rewrite a label, letting the rest of the label untouched. A very similar idea was presented in [PEM87], interestingly using orders with reversed direction compared to ours. Although more flexible wrt. the orders they need quite elaborate application conditions while we obtain simpler results (and proofs) by using lattice theory [Bir67] and are able to use non-injective morphisms in rules. Note that it is also possible to combine a rewriting formalism for the nets structure and with one for the labels, but this will result in higher complexity.

In the next section we develop two rewriting formalisms based on the so-called DPO approach. The first one is a straightforward extension of DPO to our morphisms, while the other prefers deletion of (parts of) labels, in the case of a conflict. In Section 3 and 4 we show how our approach can be used to realize adaptive user interfaces based on XML inscriptions. Due to space restrictions the proofs are published only in a long version [SW14] of this paper.

2 Coloured Petri Net Rewriting with Lattices

Since our focus lies on the rewriting formalism, we consider Petri nets as special kinds of graphs which can be transformed as described in [Roz97]. We will not define the semantics of coloured Petri nets [Jen97] and just assumes the existence of inscriptions of transitions, places, and arcs. In practice these inscriptions are often used to model guard conditions or typing tokens.

Definition 1 (*In*-Coloured Petri Net) An *In-coloured Petri net* is a 6-tuple (P,T,E,In,c,in) with P a set of *places*, T a set of *transitions*, and E a set of *edges* where P, T, and E are pairwise disjoint. The function $c : E \to (P \times T) \cup (T \times P)$ defines the source and target of each edge. In is a (possibly infinite) set of inscriptions and the total function $in : (P \cup T \cup E) \to In$ assigns an inscription to each element of the net.

Often, transformation formalisms can only change inscriptions by deleting and recreating the corresponding objects, since morphisms are usually required to preserve inscriptions. Exceptions are for instance [HP02, Ros75], where labelling functions can be partial and [PEM87], which uses orders on inscriptions. We pursue the latter approach by extending the notion of morphisms with orders and using lattices [Bir67] to order inscriptions.

Definition 2 (Total $\sqsubseteq$-Morphisms on *In*-Coloured Petri Nets) Let $\sqsubseteq$ be a partial order on the set of inscriptions In. A *total $\sqsubseteq$-morphism* $r : A \to B$ on coloured Petri nets $A = (P_A, T_A, E_A, In, c_A, in_A)$

and $B = (P_B, T_B, E_B, In, c_B, in_B)$ is a triple (r_P, r_T, r_E) of the three total morphisms $r_P : P_A \to P_B$, $r_T : T_A \to T_B$, and $r_E : E_A \to E_B$, such that the following conditions hold (omitting indices or r):

$$\forall e \in E_A \text{ with } c_A(e) = (x,y) : c_B(r(e)) = (r(x), r(y)) \text{ and}$$
$$\forall x \in (P_A \cup T_A \cup E_A) : in_A(x) \sqsubseteq in_B(r(x))$$

A $\sqsubseteq$-morphism is an isomorphism if it is injective, surjective and inscription preserving, i.e. in the second condition above equality holds.

Definition 3 (Complete lattice) A *complete lattice* is a pair $(L, \sqsubseteq)$, where L is a set and $\sqsubseteq$ is a partial order on L. Furthermore, for every set $L' \subseteq L$ there is an infimum (greatest lower bound) $\sqcap L' \in L$ such that: 1) for all $l \in L'$, $\sqcap L' \sqsubseteq l$ holds, and 2) for all l' satisfying the first condition, if $\sqcap L' \sqsubseteq l'$, then $\sqcap L' = l'$. Analogously, there is a supremum (least upper bound) $\sqcup L' \in L$ such that: 1) for all $l \in L'$, $l \sqsubseteq \sqcup L'$ holds, and 2) for all l' satisfying the first condition, if $l' \sqsubseteq \sqcup L'$, then $\sqcup L' = l'$. As shorthand we use $l_1 \sqcup l_2$ and $l_1 \sqcap l_2$ to denote $\sqcup\{l_1, l_2\}$ and $\sqcap\{l_1, l_2\}$ respectively.

We call a lattice meet-infinite distributive if $m \sqcup (\sqcap_{l \in L'} l) = \sqcap_{l \in L'} (m \sqcup l)$ and join-infinite distributive, if $m \sqcap (\sqcup_{l \in L'} l) = \sqcup_{l \in L'} (m \sqcap l)$ holds for all $m \in L$ and $L' \subseteq L$.

Note that our definition of a morphism is equal to the inscription preserving definition, if an identity relation $\sqsubseteq$ is used. In the following our inscription sets will not be ordinary lattices, but disjoint unions of complete lattices, i.e. $(In, \sqsubseteq)$ is a partial ordered set such that there is a partition Π_L over L where $(\Pi, \sqsubseteq \cap (\Pi \times \Pi))$ is a complete lattice for every $\Pi \in \Pi_L$ and if $x \sqsubseteq y$, then there is a $\Pi \in \Pi_L$ with $\{x, y\} \subseteq \Pi$. We use $\mathcal{CPN}[In, \sqsubseteq]$ to denote the category of *In*-coloured Petri nets where *In* has this form. Note that the label preserving case is subsumed by our approach since any identity relation also forms a disjoint union of complete lattices.

The double pushout approach (DPO) is based on the notion of pushouts and pushout complements. These constructions are used to add (former) and delete (latter) elements of a net in a rewriting step.

Definition 4 (Pushouts) Given two morphisms $f : A \to B$ and $g : A \to C$, the triple $(D, g' : B \to D, f' : C \to D)$ is called a *pushout* of (f, g), if: 1) $g' \circ f = f' \circ g$, and 2) for all nets E and morphisms $f^* : C \to E$ and $g^* : B \to E$ that fulfil the former constraint, there is an unique morphism $h : D \to E$ with $h \circ g' = g^*$ and $h \circ f' = f^*$.

We call (C, g, f') the *pushout complement* of (f, g') if (D, g', f') is a pushout of (f, g).

An example of a pushout can be seen in Figure 1, where the morphisms are indicated by position with the exception of the two places in A which are non-injectively mapped to the same place in B. The pushout contains the elements of B and C, but merges elements related via A, i.e. elements are merged if they share a common preimage in A. The labels of elements in D are thereby the supremum of the labels of all their preimages in B and C. Thus, D can be seen as the smallest merging of B and C via the interface A. We state the existence of pushouts and pushout complements in our setting by the following two lemmas.

Lemma 1 *For $\sqsubseteq$-morphisms $b : A \to B$ and $c : A \to C$ the pushout exists in $\mathcal{CPN}[In, \sqsubseteq]$ and is unique up to isomorphism.*

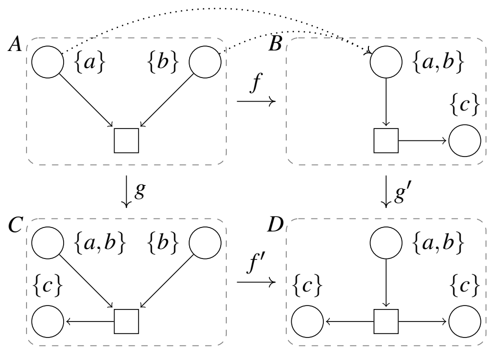


Figure 1: Example of a pushout using the lattice $(\mathcal{P}(\{a,b,c\}),\subseteq)$ as inscriptions.

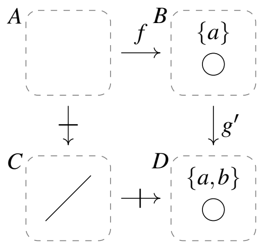


Figure 2: Example morphisms, where no pushout complement exists

Lemma 2 *For morphisms $b : A \to B$ and $d : B \to D$ the pushout complement in the category $\mathcal{CPN}[In,\sqsubseteq]$ exists, if and only if the following conditions hold:*

- *for every $x \in P_B \cup T_B$ without a preimage in A, $d(x)$ is only connected to edges with a preimage in B (dangling edge condition),*
- *for every $x,y \in P_B \cup T_B \cup E_B$, if $d(x) = d(y)$ and $x \neq y$, then x and y have preimages in A (identification condition), and*
- *for every $x \in P_B \cup T_B \cup E_B$ without a preimage in A, $in_B(x) = in_D(d(x))$ holds (inscription condition).*

The first two conditions of Lemma 2 are well-known conditions for the existence of pushout complements for general graphs. The last condition is illustrated in Figure 2. The place in D cannot have a preimage in C, since the pushout of B and C would contain two places, but then D is not minimal since the inscription would have to be $\{a\}$. Thus, no pushout complement exists in this case.

Note that pushout complements in $\mathcal{CPN}[In,\sqsubseteq]$ are not necessarily unique, even if all involved morphisms are injective. We approach this ambiguity by introducing the notions of preservation-focused and deletion-focused rewriting. While the first one arises from a natural refinement of DPO rewriting, the latter notion prefers deletion to preservation when rewriting inscriptions and its application is illustrated in more detail in Section 3.

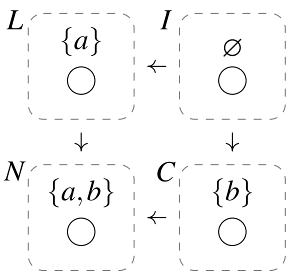
Definition 5 (DPO Rule and Matching) A *(DPO) $\sqsubseteq$-rule* ρ is a pair of $\sqsubseteq$-morphisms $l : I \to L$ and $r : I \to R$. A *$\sqsubseteq$-match* of a rule ρ to a net N is a $\sqsubseteq$-morphism $m : L \to N$.

Definition 6 (Preservation-Focused Rewriting) Let $l : I \to L$ and $r : I \to R$ be a rule and let $m : L \to N$ be a match of the rule in N. A net N can be rewritten to a net N' if there is a minimal pushout complement C, $m' : I \to C$, $l' : C \to N$ such that N' is isomorphic to the pushout of m' and r. A pushout complement C is minimal if for all pushout complements D, $m'' : I \to D$, $l'' : D \to N$ it holds that if there exists an injective $\sqsubseteq$-morphism $k : D \to C$ with $m' = k \circ m''$ and $l'' = l' \circ k$, then k is an isomorphism.

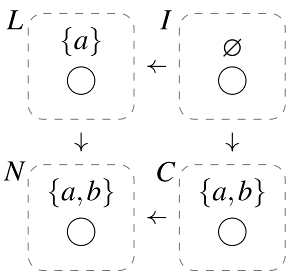
We call a rule preservation-applicable, if at least one pushout complement exists.

When using preservation-focused rewriting, inscriptions are rewritten in the classical DPO sense. A rule application tries to delete the ”difference” between an inscription in L and its preimage in I from its image in N. If this deletion is not possible, the deletion is not performed. However, the rule is still preservation-applicable (taking the conditions of Lemma 2 into account), but the inscription remains unchanged, as demonstrated in the following example.

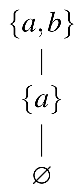
Example 1 *Figure 3a shows the minimal pushout complement using the lattice* $(\mathcal{P}(\{a,b\}),\subseteq)$*, since* $\{b\}$ *is the smallest inscription such that* $\{b\} \sqcup \{a\} = \{a,b\}$*. The rule can simply delete the 'a' part of the inscription* $\{a,b\}$ *to obtain* $\{b\}$*. However, this is not always possible as shown in Figure 3b, where the lattice in Figure 3c is used. This lattice does not contain* $\{b\}$ *such that* $\{a,b\}$ *is now the minimal inscription. Effectively the inscription remains unchanged although the rule specifies a (partial) deletion. Note that Figure 3b is also a pushout complement if* $(\mathcal{P}(\{a,b\}),\subseteq)$ *is used, although it is not minimal in that case.*



(a) The minimal pushout complement using the lattice $(\mathcal{P}(\{a,b\}),\subseteq)$



(b) The minimal pushout complement using the lattice shown in Figure 3c



(c) Hasse diagram of a lattice

Figure 3: A preservation-focused rule application using two different (distributive) lattices

In general a preservation-focused rewriting step generates a set of rewritten nets, but we can state the following uniqueness criterion.

Proposition 1 *Let* ρ *be a* $\sqsubseteq$*-rule where the left leg is injective and let* m *be a* $\sqsubseteq$*-match such that* ρ *is preservation-applicable to some net* N*. The preservation-focused application of* ρ *to* N *via* m *results in a unique net* N' *(up to isomorphism) if the set of inscriptions of* N *is a disjoint union of lattices and each lattice is meet-infinite distributive.*

In preservation-focused rewriting, places, transitions and arcs with larger inscriptions than their preimage in L cannot be deleted due to the third condition of Lemma 2 and inscriptions may remain unchanged even if the rule specifies otherwise. These properties are not always desired, for instance not in our application presented in Section 3 and 4. Therefore, we present an alternative approach which deletes as much of the inscriptions as necessary in these situations. As an auxiliary construction we define a functor mapping form $\mathcal{CPN}[In,\sqsubseteq]$ to $\mathcal{CPN}[In,=]$, the category where $\sqsubseteq$ is the identity. The functor effectively removes the inscriptions.

Definition 7 (Forgetful functor) Let $F : \mathcal{CPN}[In,\sqsubseteq] \to \mathcal{CPN}[In,=]$ be a functor. For every object $A = (P_A, T_A, E_A, In, c_A, in_A)$ of $\mathcal{CPN}[In,\sqsubseteq]$ we define $F(A) = (P_A, T_A, E_A, In, c_A, in'_A)$ with $in'_A(x) = \bigsqcap \Pi$ for all $x \in P_A \cup T_A \cup E_A$, where $in_A(x) \in \Pi$ for some element Π of the partition Π_{In} of In. For every arrow $m : A \to B$ we define $F(m)(x) = m(x)$ for all $x \in P_A \cup T_A \cup E_A$.

Definition 8 (Deletion-Focused Rewriting) Let $l : I \to L$ and $r : I \to R$ be a rule and let $m : L \to N$ be a match of the rule in N. A deletion-focused rewriting step is performed in the following way:

1. Calculate a pushout complement $N' = (P_{N'}, T_{N'}, E_{N'}, In, c_{N'}, in_{N'})$ of $F(l)$ and $F(m)$ with morphisms $m' : F(I) \to N'$, $l' : N' \to F(N)$.

2. For every $x \in N'$ let the set of inscriptions $\mathcal{I}_x$ be defined as follows:

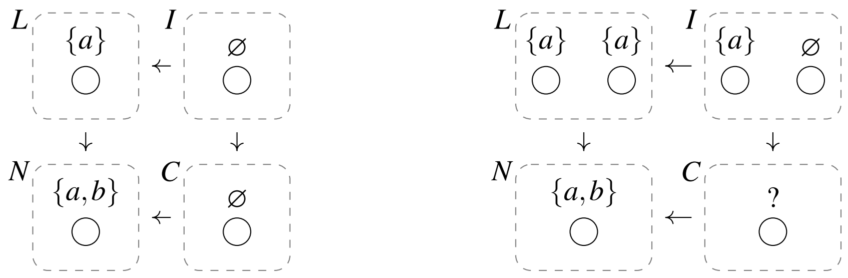
$$\mathcal{I}_x = \{z \in In \mid (\forall x' \in I : (m'(x') = x \Rightarrow z \sqcap in_L(l(x')) = in_I(x'))) \wedge z \sqsubseteq in_N(l'(x))\}.$$

   If $\mathcal{I}_x$ is non-empty for all $x \in N'$, construct a net $N'' = (P_{N'}, T_{N'}, E_{N'}, In, c_{N'}, in_{N''})$ where $in_{N''}(x)$ is any maximal element of $\mathcal{I}_x$ and the morphisms $m'' : I \to N''$, $l'' : N'' \to N$ with $m''(x) = m'(x)$, $l''(x) = l'(x)$.

3. Calculate the pushout of m'' and r to obtain the rewritten Petri net M.

We call a rule deletion-applicable if the first two conditions of Lemma 2 hold and for at least one net N' calculated in the first step, $\mathcal{I}_x$ is non-empty for all $x \in N'$.

By construction m'' and l'' defined in Definition 8 are valid $\sqsubseteq$-morphisms and the diagram $l'' \circ m'' = m \circ l$ commutes, but is not necessarily a pushout. The application condition differs from preservation-focused rewriting and arises from conflicts shown in Figure 4b.



(a) Deletion-focused rewriting of the example in Figure 3b

(b) Example of a rule which is not deleting applicable

Figure 4: Examples of deletion-focused rewriting steps

Example 2 *Figure 4a shows the deletion-focused rewriting step applied to the example in Figure 3b using the lattice in Figure 3c. We search for every inscription z for which $z \sqcap \{a\} = \varnothing$ holds and which is also smaller or equal to $\{a,b\}$. Since $\varnothing$ is the only possibility, it is a maximal element. Effectively, since there is no inscription containing b without a, the deletion of a from $\{a,b\}$ also deletes b. Figure 4b shows a conflict, such that the rule is not deletion-applicable. The* *rule specifies a preservation and a rewriting of the same inscription, such that $\mathcal{I}_x$ is empty for the node in C. The node cannot be labelled with $\varnothing$ since then m'' would not be a valid $\sqsubseteq$-morphism. Note that in this case a pushout complement does exist.*

Although ambiguous in the general case, we can state a uniqueness criterion for deletion-focused rewriting which is analogue to Proposition 1.

Proposition 2 *Let ρ be a $\sqsubseteq$-rule where the left leg is injective and let m be a $\sqsubseteq$-match such that ρ is deletion-applicable to some net N. The deletion-focused application of ρ to N via m results in a unique net N' (up to isomorphism) if the set of inscriptions of N is a disjoint union of lattices and each lattice is join-infinite distributive.*

3 Deletion-focused PNML rewriting

The Petri Net Markup Language (PNML) is a well established XML-based format for making Petri net-based models persistent [HKK⁺09]. Therefore, we use PNML in our implementation as basis for describing so-called reference nets [Kum02]. These are coloured Petri nets where the tokens are references to objects in a class hierarchy and also support code execution when firing transitions. We clarify how rewriting steps will be performed in this setting, by defining a mathematical model for XML below. This also illustrates how our rewriting formalisms can be implemented in practice.

In this setting, inscriptions are XML nodes, and hence have a tree-like structure. We will show that they form a disjoint union of lattices compatible with our rewriting formalisms. We assume that XML nodes are distinguishable by an ID (which can be the node name or a designated attribute) and the order on child nodes (but not the content of nodes) is negligible. Furthermore, every XML node has a value (possibly a tuple) which describes its properties, such as attributes, excluding child nodes. We use $\uplus$ to denote the disjoint union.

Definition 9 (XML Inscription) Let $(Val, \leq)$ be a disjoint union of complete lattices Val_i of values with $\biguplus_{i\in I} Val_i = Val$ and let N be a set of IDs, which is sorted such that it can be partitioned in N_i with $N = \biguplus_{i\in I} N_i$. An XML inscription $xml_{N,Val}$ is a directed rooted tree (V, E, r, γ) of finite height, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $r \in V$ is the root and $\gamma : V \to \bigcup_{i\in I}(N_i \times Val_i)$ maps properties to each vertex. Additionally for every two edges $(v_1, v_2), (v_1, v_3) \in E$ with $\gamma(v_i) = (n_i, w_i)$ (for $i \in \{2,3\}$) it holds that $n_2 \neq n_3$.

For every $v \in V$ we define $v{\downarrow} = (V', E', v, \gamma')$ to be the subtree of $xml_{N,Val}$ with root v, which is an XML inscription itself.

Definition 10 Let $XML_{N,Val}$ be the set of all XML inscriptions $xml_{N,Val}$. We define the ordered set $(XML_{N,Val}, \sqsubseteq)$, where for two elements $(V_1, E_1, r_1, \gamma_1) \sqsubseteq (V_2, E_2, r_2, \gamma_2)$ holds if and only if: let $\gamma_i(r_i) = (n_i, w_i)$ for $i \in \{1,2\}$, then $n_1 = n_2$, $w_1 \leq w_2$, and for all $v_1 \in V_1$ with $(r_1, v_1) \in E_1$ there is a $v_2 \in V_2$ with $(r_2, v_2) \in E_2$ such that $v_1{\downarrow} \sqsubseteq v_2{\downarrow}$.

Lemma 3 $(XML_{N,Val}, \sqsubseteq)$ *is a disjoint union of complete lattices, provided Val is.*

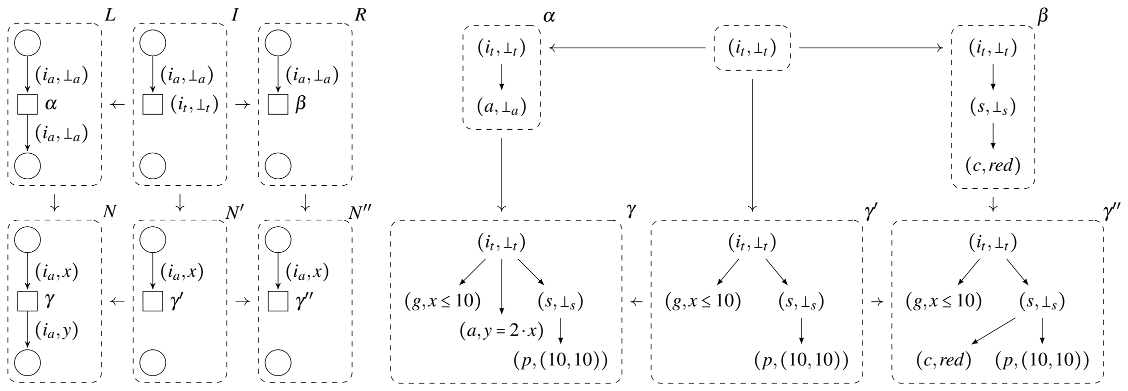
The proof of Lemma 3 is based on the following observations: each lattice In within $XML_{N,Val}$ consists of all inscriptions, where the ID of the root elements are equal. The supremum exists if and only if this is the case and can be computed inductively as follows. Let $L \subseteq In$ be a non-empty subset of In where root elements have ID k. For every ID $n \in N$, we define $C_n^{\sqcup}(L) = \{v\!\downarrow \mid (V,E,r,\gamma) \in L, (r,v) \in E, \gamma(v) = (n,w)\}$, the set of all direct subinscriptions of inscriptions of L, where the root ID is n. Furthermore, let $M = \{n \in N \mid C_n^{\sqcup}(L) \neq \varnothing\}$ be the set of all IDs for which child nodes exist and let $(V_m, E_m, r_m, \gamma_m) = \bigsqcup C_m^{\sqcup}(L)$ be their supremum for each $m \in M$. The supremum of L can be expressed as follows:

$$\bigsqcup L = (\{x\} \uplus \biguplus_{m\in M} V_m, \{(x,v) \mid v \in \{r_m \mid m \in M\}\} \uplus \biguplus_{m \in M} E_m, x, \gamma'),$$

where $\gamma'(y) = \gamma_m(y)$ for $y \in V_m$ and $\gamma'(x) = (k, \bigvee_{l\in L} w_l)$ with $l = (V_l, E_l, r_l, \gamma_l)$ and $\gamma_l(r_l) = (k, w_l)$.

The infimum can be expressed in an analogous way, with the exception that $C_n^{\sqcap}(L) = C_n^{\sqcup}(L)$ if *every* inscription of L has a child with ID n and $C_n^{\sqcap} = \varnothing$ otherwise.

Example 3 *To illustrate the use of $(XML_{N,Val}, \sqsubseteq)$, we give a complete deletion-focused rewriting step in Figure 5. The rule as well as the rewritten nets are show in Figure 5a. For clarity, the inscriptions of the shown transitions are displayed separately in Figure 5b. Inscriptions of arcs have a root ID i_a and either a variable (x or y) or no variable ($\bot_a$ with $x, y \sqsupseteq \bot_a$) as value. Root elements of inscriptions of transitions have always the ID i_t and value $\bot_t$, but have a more complex substructure. This can consist of a guard condition g (a boolean expression preventing firing), an action a (assigning the result of an arithmetic to a variable when firing) and a style s (describing the visual appearance). The style can consist of a position p and a colour c.*



(a) Deletion-focused rewriting using $(XML_{N,Val}, \sqsubseteq)$ as inscriptions

(b) Shows how the inscriptions of the transitions in Figure 5a are rewritten

Figure 5: Example of a deletion-focused rewriting of an $XML_{N,Val}$-coloured Petri net

The given rule can be matched to any net N that contains a transition with one incoming arc, one outgoing arc, and an action with any value. In the first step, it deletes the outgoing arc and the action to generate the net N'. The inscription γ' is the largest inscription satisfying $\gamma' \sqcap \alpha = (i_t, \bot_t)$ as well as $\gamma' \sqsubseteq \gamma$, since the existence of an action in γ' would cause $\gamma' \sqcap \alpha$ to contain an action as *well. Note that N' is not a pushout complement (which does not exist), since the inscription of the deleted arc is strictly larger than its preimage in L, thus, the rule is not preservation-applicable to N. In the second step, the pushout N'' is generated by calculating the supremum $\gamma'' = \gamma' \sqcup \beta$, which contains all merged subinscriptions of both γ' and β. The value of i_t in γ'' is generated by the supremum of its values in γ' and β, i.e. by $\bot_t \sqcup \bot_t = \bot_t$ (the same holds for s). Effectively the transition is marked with the colour red, without changing other layout properties.*

In addition to the previous result, we can show that our approaches rewrite uniquely, if the lattices of values are meet-infinite or join-infinite distributive.

Lemma 4 *Every lattice of $(XML_{N,Val}, \sqsubseteq)$ is meet-infinite (or join-infinite) distributive, if every lattice of Val is meet-infinite (or join-infinite) distributive.*

4 Application to User Interface Reconfiguration

Adaptable UIs offer a great benefit to human-computer interaction, according to the fact that those UIs can be adapted to the user's personal preferences and abilities. The use of a formal modelling approach in this context offers the opportunity to close the gap between modelling and execution of UIs on the one hand and the implementation of adaptable UIs in a full-fledged computer-processable format on the other. Based on a two-layered representation of a UI, we developed a visual modelling language for interactive modelling of interaction logic by experts. Interaction logic can be defined as a data processing layer, modelling data-based communication between the physical representation and the system to be controlled. The physical representation is the second layer of the UI that directly interacts with the user and can be specified as a set of widgets, such as buttons, sliders, or text fields, etc. In the interaction logic, events are being processed that occur after, e.g., the user pressed a button or after he used another interaction element of the physical representation. Vice versa, data emitted from the underlying system is prepared to be presented to the user via the physical representation. Beside this data-based communication between user and system (also called business logic), also dialogue-specific structures are specified in interaction logic. Here, data-based dependencies between input events and system data can influence interaction by predefined logic conditions.

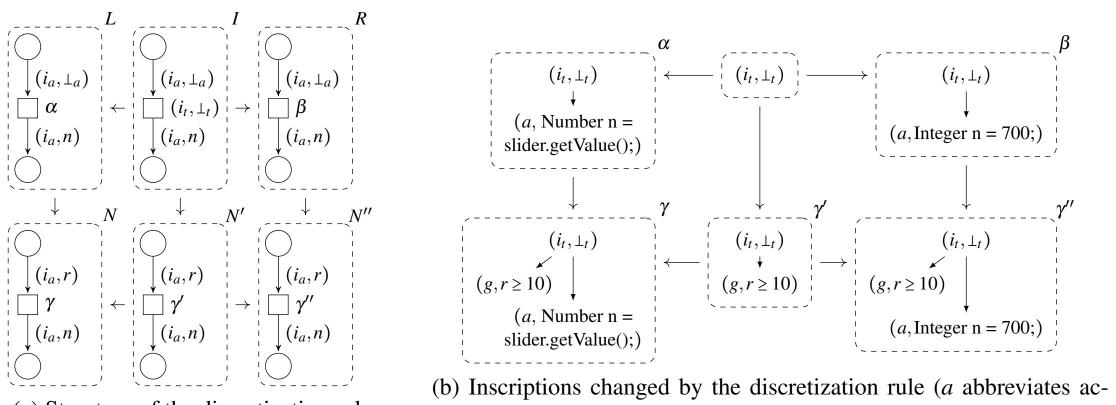
For visually modelling, a graph-based visual language called FILL [Wey12] has been developed, that is transformed into reference nets as introduced by Kummer [Kum02]. The reason for this is motivated by various aspects, such as that the transformation defines a formal semantic for FILL, reference net-based interaction logic is executable using the implemented open-source simulator RENEW [KWD], and finally, interaction logic is accessible for formal graph rewriting concepts, as described in the paper at hand. Thus, based on this transformation and the rewriting approaches introduced, the full-fledged concept required for the development of formal adaptable UIs is provided.

For modelling user interfaces using FILL, a visual and interactive editor has been developed, called UIEditor[1]. The editor is separated into two visual editors to (a) model the physical representation of a visual user interface and (b) to model interaction logic using FILL. For execution

[1] www.uieditor.org

of the user interface model, the UIEditor offers a simulation component, which is also capable to transform FILL models into reference nets. Thus, a computer parseable representation of such a reference net has to be provided. We decided to use PNML, which is the main reason for applying the lattice-extended rewriting approach to PNML in Section 3. The whole transformation algorithm has been described in [Wey12, pg. 44–84]. A third component implements interactive reconfiguration, as it will be described in more detail, below, which is responsible for interactively creating rewriting rules. This interactive creation is the major aspect of implementing adaptable user interfaces, since the engineer modelling the UI is not able to foresee all possible adaptations a user could have in mind. Hence, the changes in the UI – both in the physical representation and the interaction logic – should be controlled by the user and need not be predefined by the application provider (although this is also possible).

Using graph transformation to change interaction logic, the behaviour of a given user interface can be adapted to certain requirements. Paired with the ability of reference nets to be executed based on simulation, the changes can be directly tested and used in an application scenario. For instance, in [WBLK12] such an scenario has been described, where users were asked to reconfigure an initially given user interface of a simple simulation of a steam water reactor according to a variety of trained control tasks. These tasks were embedded to a controlling scenario of the reactor simulation. Here, the user has to start and stop the reactor, or to handle upcoming system errors, such as the blackout of a water pump. In a test run, two groups were asked to perform these tasks in a predefined test scenario [BWKL13]. The experiment group was able to interactively reconfigure the user interface by choosing from a predefined set of operations, where the control group did not have this option. The users were able to combine various buttons to one, which was able to perform all operations in parallel, that were former triggered by the selected buttons. Furthermore, users were able to discretize a continuous input operation, e.g., represented as a slider widget. For instance, a user can select a slider, chose the discretization operation, and define the discrete integer value that should be settable by a newly generated button. The rewriting rule generated for such an adaptation is shown in Figure 6, including an exemplary rule application.



(a) Structure of the discretization rule

(b) Inscriptions changed by the discretization rule (a abbreviates actions, g abbreviates guard conditions)

Figure 6: Example of the rewriting rule for the discretization, including an application

All reconfiguration operations were applied to reference net-based interaction logic using deletion-focused PNML rewriting in a two step process. In the first step, the user interactively selected the interaction elements that should be affected (e.g. the slider), as well as the reconfiguration operation (e.g. discretization) that should be applied. The second step has been implemented algorithmically and was responsible for selecting the affected graphical parts of interaction logic (e.g. the button) and generating the XML-based graph rewriting rule, containing all parts to be changed and being applied to interaction logic afterwards. In Figure 6a the resulting rule of the discretization operation can be seen where the inscriptions are show Figure 6b. Here, the lattice uses class inheritance to define a rule pattern which can match various number types in the inscription, such as Double or Integer. The new transition generates a specific integer value implementing the functionality of the new button, while the guard condition remains unchanged, since it was not specified to change by the rule.

5 Conclusion

By adding an order on the inscriptions, we introduced two rewriting formalisms for coloured Petri nets, which are also able to (partially) change inscriptions. The first formalism is a straightforward extension of the classical DPO approach, while the second formalism tries to add an SPO-like behaviour on the inscriptions, still providing the same behaviour on the net structure. The latter approach has similarities with the so called Sesqui Pushout approach introduced in [CHHK06], where the left leg of a rule is not applied by calculating the pushout complement, but the final pullback complement. The main difference is that all incident edges are cloned in SqPO, if a node is split by a rule. Further, there are rules, where our deletion-focused rewriting will be ambiguous while SqPO is not applicable due to the fact that the final pullback complement is unique if it exists. Our approach is similar to [PEM87] while correcting a minor error already mentioned in [HP02], coming from incorrect conditions for the existence of pushout complements. In Section 3 and 4 we introduced a disjoint union of lattices compatible with our formalisms and illustrated how the UIEditor uses this formalisms to realize adaptive user interfaces. I this context the approach of [LO04] could be interesting, where the application of transformation rule may depend on the current marking of the net.

Although out of the possibilities of this paper, it is not difficult to introduce typical extensions of the DPO approach into our approach, for instance negative application conditions. Furthermore, its extension to other types of labelled graphs is quite straightforward.

Acknowledgements: We thank Barbara König for the helpful discussion on lattice theory and its possible application in our setting.

Bibliography

[Bir67] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1967.

[BWKL13] D. Burkolter, B. Weyers, A. Kluge, W. Luther. Customization of user interfaces to reduce errors and enhance user acceptance. *Applied Ergonomics*, 2013.

[CHHK06] A. Corradini, T. Heindel, F. Hermann, B. König. Sesqui-pushout rewriting. In *Proc. of ICGT '06*. Pp. 30–45. Springer, 2006. LNCS 4178.

[CMN80] S. K. Card, T. P. Moran, A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of ACM* 23:396–410, 1980.

[EHP06] H. Ehrig, K. Hoffmann, J. Padberg. Transformations of Petri Nets. *Electr. Notes Theor. Comput. Sci.* 148(1):151–172, 2006.

[HJSW10] F. Heidenreich, J. Johannes, M. Seifert, C. Wende. Closing the Gap between Modelling and Java. In *Software Language Engineering, LNCS 5969*. Pp. 374–383. 2010.

[HKK$^+$09] L. M. Hillah, E. Kindler, F. Kordon, L. Pertrucci, N. Trèves. A primer on the Petri Net Markup Language and ISO/IEC 15909-2. In *Petri Net Newsletter*. Volume 76. Gesellschaft für Informatik, Bonn, 2009.

[HP02] A. Habel, D. Plump. Relabelling in Graph Transformation. In *Proc. of ICGT '02*. Pp. 135–147. Springer, 2002. LNCS 2505.

[Jen97] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol 1, Basic Concepts*. Springer, 1997.

[JWZ93] C. Janssen, A. Weisbecker, J. Ziegler. Generating user interfaces from data models and dialogue net specifications. In *Proc. of INTERACT '93 and CHI '93*. CHI '93, pp. 418–423. ACM, New York, NY, USA, 1993.

[KP85] D. Kieras, P. G. Polson. An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies* 22(4):365–394, 1985.

[Kum02] O. Kummer. *Referenznetze*. Logos, 2002.

[KWD] O. Kummer, F. Wienberg, M. Duvigneau. Renew—The Reference Net Workshop, online, URL: http://renew.de/ (last visited: 30-10-2013).

[LO04] M. Llorens, J. Oliver. Introducing Structural Dynamic Changes in Petri Nets: Marked-Controlled Reconfigurable Nets. In Wang (ed.), *Automated Technology for Verification and Analysis*. LNCS 3299, pp. 310–323. Springer Berlin, 2004.

[NPLB09] D. Navarre, P. Palanque, J.-F. Ladry, E. Barboni. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM TOCHI* 16(4):1–56, 2009.

[PEM87] F. Parisi-Presicce, H. Ehrig, U. Montanari. Graph rewriting with unification and composition. In *Proceedings of the 3rd Int'l Workshop on Graph-Grammars and Their Application to Computer Science*. Pp. 496–514. Springer, 1987.

[Ros75] B. K. Rosen. Deriving Graphs from Graphs by Applying a Production. *Acta Informatica* 4:337–357, 1975.

[Roz97] G. Rozenberg (ed.). *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing, 1997.

[SW14] J. Stückrath, B. Weyers. Lattice-extended Coloured Petri Net Rewriting for Adaptable User Interface Models. Technical report 2014-01, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, 2014.

[WBLK12] B. Weyers, D. Burkolter, W. Luther, A. Kluge. Formal modeling and reconfiguration of user interfaces for reduction of errors in failure handling of complex systems. *Human-Computer Interaction* 28(10):646–665, 2012.

[Wey12] B. Weyers. *Reconfiguration of User Interface Models for Monitoring and Control of Human-Computer Systems*. Dr. Hut, 2012.