Petri nets with may/must semantics: Preserving properties through data refinements

Petri Nets with May/Must Semantics — Preserving Properties through Data Refinements

Olga Kouchnarenko[1], Natalia Sidorova[2], and Nikola Trčka[2]

[1] LIFC and INRIA/CASSIS
University of Franche-Comté
16 route de Gray 25030 Besançon CEDEX, France
Olga.Kouchnarenko@loria.fr
[2] Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
n.sidorova@tue.nl, n.trcka@tue.nl

Abstract. Many systems used in process managements, like workflow systems, are developed in a top-down fashion, when the original design is refined at each step bringing it closer to the underlying reality. Underdefined specifications cannot however be used for verification, since both false positives and false negatives can be reported. In this paper we introduce colored Petri nets where guards can be evaluated to true, false and *indefinite* values, the last ones reflecting underspecification. This results in the semantics of Petri nets with may- and must-enableness and firings. In this framework we introduce property-preserving refinements that allow for verification in an early design phase. We present results on property preservation through refinements. We also apply our framework to workflow nets, introduce notions of may- and must-soundness and show that they are preserved through refinements. We shortly describe a prototype under implementation.
Keywords: Petri nets; workflow; refinement, may-/must-soundness; property preservation.

1 Introduction

There is much to be gained from a good understanding and a simple description while writing formal specifications. In the refinement-based development, the basic idea is to introduce new details to complement specifications. For example, the engineer might be more precise about the way data should be interpreted, or the way certain computations are to be carried out. Thus, refinements in general result in complicating the system.

The process of specification refinement involves the removal of non-determinism or uncertainty. An abstract specification may leave design choices unresolved while in a refinement some of these choices are resolved. Several refinement steps may be performed, each removing another degree of uncertainty, until the specification reaches the required format.

The main principle of refinement methods is that if the initial abstract specification is correct and refinement steps preserve correctness, then the resulting specification (or even implementation) will be correct by construction. The errors can therefore be discovered in the early phases of the design. Moreover, since an abstract program is, in general, easier to prove correct than concrete one, the refinement approach simplifies the structuring of the verification process.

In this paper we consider Coloured Petri nets [15, 16] as the modeling language used in the refinement-based design. In Coloured Petri nets, a state of a net, called a *marking*, is a multiset of tokens that reside on places and carry data values; transition firings are conditioned by data-dependent guards, consume input tokens and compute output tokens whose values depend on the values of the input ones. Coloured Petri nets are widely used for modeling in many application domains like manufacturing, workflow management, control systems, etc. (see [10] for an extensive list of industrial applications of Coloured Petri nets). Moreover, a number of industrial tools for designing coordination layers, and in particular workflows systems, are Petri net-based [2].

Coloured Petri nets are usually developed in a top-down fashion, when the original design is refined at each step bringing it closer to underlying reality. The design normally starts with defining the basic control structure, and only later data is added to the model step by step. In the current verification practice, underdefined specifications are often verified as if no data is involved in the making of choices [1]. This can lead to obtaining both false positives and false negatives [23], and thus the added value of the verification effort is questionable.

In this paper we introduce a possibility to indicate underdefined pieces of a specification by 1) allowing the use of the indefinite value $\top$ (unknown) in transition guards, and 2) supporting data-type refinements. This induces a semantics of Petri nets with may- and must-enabledness of transitions while maintaining the standard may Petri nets semantics. This semantics combines well with refinements: a transition is may-enabled in a marking m of net N if it may be enabled in some refinement of (N, m), and a transition is must-enabled in a marking m of net N if it is enabled in any refinement of (N, m). We show how properties like deadlock freeness or livelock freeness are preserved trough refinements from abstract systems to refined systems.

We pay a particular attention to Workflow nets [3] – Petri nets modeling workflows. A Petri net is a workflow net iff it possesses one place without incoming arcs (initial place), one place without outgoing arcs (final place) and all other places and transitions lie on paths from the initial to the final place. A process execution starts in a workflow net from the initial marking consisting of a single token on the initial place. In a properly designed workflow net, any process execution leads to the final marking consisting of a single token on the final place. This property is called *soundness* [3, 12]. Note that soundness implies the absence of deadlocks (the final marking excluded) and livelocks (infinite cycles are allowed but they can be removed under the global fairness assumption [11]). We adapt the notion of soundness to our net by introducing may-soundness and must-soundness, where not may-soundness implies that any refinement of the

net is unsound (proving the design wrong), and must-soundness implies that any refinement of the net is sound (proving the design correct).

Related questions on the preservation temporal properties through refinements have been addressed in the context of 2-valued semantics (see, e.g., [18] or [20]). The focus there is however different. Our motivation lies in the incompleteness of specifications under verification, and we need to reason using the 3-valued semantics where a property/formula can be evaluated to *true, false* or *indefinite*. Then, when the value of the formula in the abstract model is indefinite, the refinement may bring new details and the formula value in the concrete model may become *true* or *false*, or may remain *indefinite*.

The rest of the paper is organised as follows. In Section 2 we give a motivating example illustrating the intuition behind our approach. Section 3 sketches the basic definitions needed. Section 4 introduces the definitions related to colored Petri nets and colored Workflow nets, and the notion of refinement. In Section 5 a link between our refinement notion and data refinements is established. Section 6 briefly reports on a prototype. Section 7 concludes the paper with an assessment of achievements and a discussion of future work.

2 Motivating Example

To illustrate specification refinement we consider a simplified description of a business process at an apartment letting agency. [1]

A client contacts the agency to rent an apartment he/she liked. The agency asks him/her to bring the salary slips. If the salary is considered to be high enough to rent this apartment, the agency draws up the tenancy agreement. Otherwise the client has to find a guarantor with a regular income, whose credit history and income is checked then by an independent agency. Should the guarantor fail the checks, the client can ask someone else to be a guarantor. If the guarantor is found to be reliable, the agency draws up a tenancy agreement.

A model (here a Coloured Petri net) is usually created as a graphical drawing as shown in Fig. 1. The model contains six places (drawn as ellipse or circles), eight transitions (drawn as rectangular boxes), a number of directed arcs connecting places and transitions and finally some textual inscriptions next to the places, transitions and arcs. The two left most places and a transition model the system to take the rent and salary information. By convention, the names of the places are written inside the ellipses. The inscriptions are written in the Coloured Petri net ML language which is an extension of the Standard ML language.

Transitions *insufficient salary* and *sufficient salary* have $\top$ as a guard, meaning that the condition for taking one or the other is to be specified later. Similar thing holds for the other data-dependant choices. The other transitions do not have guards, meaning that their guards are always true. Note that this workflow can be considered fully abstract, both in terms of data types (only the simple UNIT type is used) and guards (all important guards are $\top$).

[1] http://www.your-move.co.uk/lettings/tenants/student_information.htm

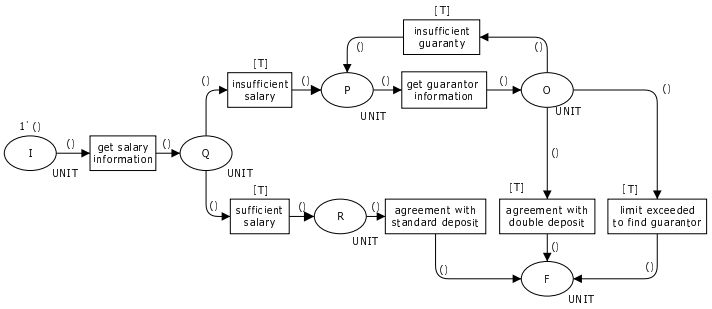

Fig. 1. Example of an abstract model

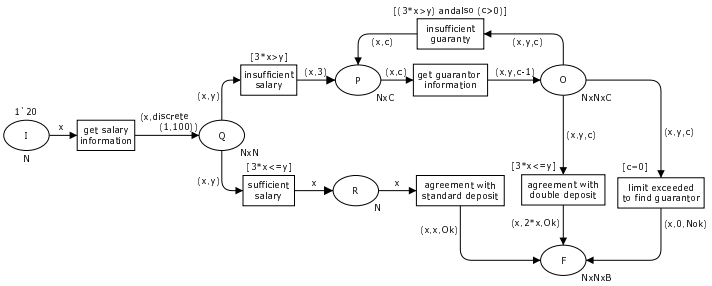

Fig. 2. Example of a refined model

Refinement We now incorporate some additional information into our first model to obtain the actual executable workflow. The client's monthly salary (randomly generated number between 1 and 100 and stored in variable y) is considered to be sufficient if it is greater than three times the rent (x - initialized to 20 in the initial marking). The agency restricts the number of attempts to find a guarantor by three attempts (counter c). Moreover, the tenancy agreement includes a clause obliging to put a one-month rent to a blocked bank account as security deposit for clients with sufficient income and two-month rent for clients who have a guarantor. Figure 2 displays the refined workflow net.

3 Basic Notions

Let P be a set. A *bag (multiset)* m over P is a mapping $m : P \to \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers. We denote the set of all bags over P by μP. We use $+$ and $-$ for the sum and the difference of two bags and $=, <, >, \leq, \geq$ for comparison of bags, which are defined in a standard way. For example, we write $m = 2[p]+[q]$ for a bag m with $m(p) = 2$, $m(q) = 1$, and $m(x) = 0$ for $x \notin \{p, q\}$.

We overload the set notation, writing $\emptyset$ for the empty bag and $\in$ for the element inclusion. As usual, $|m|$ stands for the number of elements in bag m.

Data types and operations Let Σ be a non-empty set of *data-types*, where each data type is a set of *data-values*. Let Var be a set of variables, and let type : $Var \to \Sigma$ be a function assigning a type to every variable. We assume a set $Expr$ of (well-typed) expressions built over values and variables, and we assume that type has been lifted to expressions in the standard way. For $e \in Expr$, $Var(e)$ denotes the set of (free) variables appearing in e.

We define a type $3Bool \subseteq \Sigma$ as a set $\{true, false, \top\}$ together with the truth non-strict ordering relation $\ll$ satisfying $false \ll \top \ll true$. On $3Bool$ we define the unary operation $\neg$ as $\neg false = true$, $\neg true = false$ and $\neg\top = \top$, and we define two binary operations $\wedge$ and $\vee$ as the minimum, resp. the maximum, interpreted with respect to $\ll$. The set of expressions built over $3Bool$ is denoted $Expr_{3Bool}$.

4 Colored Petri Nets with 3-valued Guards

We take a slight modification of the classical definition of Coloured Petri nets from [14] and the definition from [19]. The main difference is that the guards are interpreted w.r.t. the 3-valued semantics, i.e., they evaluate to either *true*, *false* or $\top$. We, moreover, do not allow for expressions on incoming arcs.

Definition 1 (Coloured Petri Net). *A* Coloured Petri Net (CPN) *is a tuple* $N = \langle P, T, \mathcal{A}, C, \mathcal{E}, G\rangle$ *where:*

- P *is a set of* places*;*
- T *is a set of* transitions*, with* $P \cap T = \emptyset$*;*
- $\mathcal{A}$ *is a set of* arcs*, with* $\mathcal{A} \subseteq P \times T \cup T \times P$*;*
- C *are* colors *of places, i.e.* $C : P \to \Sigma$*;*
- $\mathcal{E} : \mathcal{A} \to Expr$ *is the set of* arc inscriptions *such that*
  1. *if* $(p,t) \in \mathcal{A}$*, then* $\mathcal{E}(p,t) \in Var$ *and* $\text{type}(\mathcal{E}(p,t)) = C(p)$*; and*
  2. *if* $(t,p) \in \mathcal{A}$*, then* $\text{type}(\mathcal{E}(t,p)) = C(p)$ *and* $Var(\mathcal{E}(t,p)) \subseteq \bigcup_{(p,t)\in\mathcal{A}} Var(p,t)$.
- $G : T \to Expr_{3Bool}$ *is a* guard function *satisfying* $Var(G(t)) \subseteq \bigcup_{(p,t)\in\mathcal{A}} Var(p,t)$.

Note that there is at most one arc in each direction for any element in $P \times T$. Note also that, without loss of expressivity, we disallow the same variable name to appear on arcs having different types of their input places.

Given a node $x \in P \cup T$, the *preset* ${}^\bullet x$ of x is defined as $\{y \mid (y,x) \in A\}$ and the *postset* $x^\bullet$ is $\{y \mid (x,y) \in A\}$. We will say that a node n is a *source* node iff ${}^\bullet n = \emptyset$ and n is a *sink* node iff $n^\bullet = \emptyset$.

The state of a CPN is defined by its *marking* which is a bag over the set $\{(p,c) | p \in P, c \in C(p)\}$. The set $\mathfrak{M}$ of all possible markings is thus $\mu\{(p,c) \mid p \in P, c \in C(p)\}$. A pair (N,m) is called a *marked* CPN. The set of colors of tokens on place p in marking m is denoted by $m(p)$, i.e. $m(p) \stackrel{\text{def}}{=} \{c \mid (p,c) \in m\}$.

A *binding* is a mapping b: $Var \to \bigcup_{\sigma \in \Sigma} \sigma$ such that $b(v) \in \text{type}(v)$. We denote the set of all bindings by $\mathcal{B}$. A pair (t, b), where $t \in T$ and $b \in \mathcal{B}$ is called a *binding element*. For an expression e, $b(e)$ denotes the value of e when every $v \in Var(e)$ is replaced by $b(v)$. A binding $b \in \mathcal{B}$ is *relevant* with respect to a marking m and a transition $t \in T$ if for all $p \in {}^\bullet t$ and $v \in Var(\mathcal{E}(p,t))$ we have $b(v) \subseteq m(p)$, which means that m contains tokens needed for the binding. The set of all bindings relevant with respect to m, t is denoted by $\mathcal{B}(m,t)$.

Working with the three-valued logic we introduce a *may-must* semantics [21] for Petri nets, i.e. consider two kinds of transitions: *may*-transitions, that are possibly present, and *must*-transitions that for sure exist.

A binding element is *may-enabled* in m if $b \in \mathcal{B}(m,t)$ and $false \neq b(G(t))$. This means that there are enough input tokens of the right type and that the guard *might* be true. Similarly, (t, b) , is *must-enabled* in m if $b \in \mathcal{B}(m,t)$ and $b(G(t)) = true$, i.e. the guard *is* true. An enabled (t, b) (may or must) can *fire* leading to the marking m' defined by: $m' = m - \sum_{p \in {}^\bullet t}(p, b(\mathcal{E}(p,t))) + \sum_{p \in t^\bullet}(p, b(\mathcal{E}(t,p)))$. Depending whether t is may- or must-enabled we denote this firing by $m \xrightarrow{(t,b)}_{\text{may}} m'$ or $m \xrightarrow{(t,b)}_{\text{must}} m'$. We write $m \xrightarrow{t}_{\text{may}} m'$, resp. $m \xrightarrow{t}_{\text{must}} m'$, when there is a $b \in \mathcal{B}(m,t)$ such that $m \xrightarrow{(t,b)}_{\text{may}} m'$, resp. $m \xrightarrow{(t,b)}_{\text{must}} m'$. We write $m \longrightarrow_{\text{may}} m'$, resp. $m \longrightarrow_{\text{must}} m'$ when there is a $t \in T$ such that $m \xrightarrow{t}_{\text{may}} m'$, resp. $m \xrightarrow{t}_{\text{must}} m'$. We use $\xrightarrow{*}_{\text{may}}$ and $\xrightarrow{*}_{\text{must}}$ to denote the reflexive-transitive closure of $\longrightarrow_{\text{may}}$ and $\longrightarrow_{\text{must}}$ respectively.

The following definition introduces a notion of (behavioral) refinement for Coloured petri nets; it is inspired by the refinement notion of [21] and adapted to our framework.

Definition 2 (Refinement). *Let N_1 and N_2 be two CPNs. A relation $R \subseteq \mathfrak{M}(N_1) \times \mathfrak{M}(N_2)$ is called a* refinement *if, for every $(m_1, m_2) \in R$, the following holds:*

1. *if $m_1 \xrightarrow{t}_{must} m_1'$ for some $m_1' \in \mathfrak{M}(N_1)$, then there exist an $m_2' \in \mathfrak{M}(N_2)$ such that $m_2 \xrightarrow{t}_{must} m_2'$ and $(m_1', m_2') \in R$; and*
2. *if $m_2 \xrightarrow{t}_{may} m_2'$ for some $m_2' \in \mathfrak{M}(N_2)$, then there exist an $m_1' \in \mathfrak{M}(N_1)$ such that $m_1 \xrightarrow{t}_{may} m_1'$ and $(m_1', m_2') \in R$.*

For two marked CPNs (N_1, m_1) and (N_2, m_2) we write $(N_2, m_2) \preccurlyeq (N_1, m_1)$ if there is a refinement R such that $(m_1, m_2) \in R$.

Coloured Workflow nets In this paper we particularly focus on the *Workflow nets (WF-nets)* [3]. As the name suggests, WF-nets are used to model the processing of tasks in workflow processes. The initial and final nodes indicate respectively the initial and final states of processed cases. We add colors to WF-nets and obtain Coloured WF-nets (CWF-nets).

Definition 3 (Coloured Workflow nets). *A Coloured Petri net N is a* Coloured Workflow net (CWF-net) *iff:*

1. *It has two special places: i and f. The initial place i is a source place, i.e. ${}^\bullet i = \emptyset$, and the final place f is a sink place, i.e. $f^\bullet = \emptyset$.*
2. *For any node $n \in (P \cup T)$ there exists a path from i to n and a path from n to f along the arcs of the net.*

One natural correctness requirement for WF-nets is *soundness* [3] which requires *proper termination* for every marking reachable from the initial marking and the absence of dead transitions. We adapt this notion for the coloured case by requiring that an arbitrarily colored initial marking *terminates properly* and that for any transition t there is a (colored) initial marking $[(i, c)]$ such that t is not dead in m.

Definition 4 (May/Must Soundness for CWF-nets). *A WF-net N is* may-sound *iff the following two conditions hold:*

- *for all $c \in C(i)$ and $m \in \mathfrak{M}(N)$ such that $[(i, c)] \xrightarrow{*}_{must} m$, we have $m \xrightarrow{*}_{may} [(f, c_1)]$ for some $c_1 \in C(f)$, and*
- *for every $t \in T$ there exists a color $c \in C(i)$ and two markings m, m' such that $[(i, c)] \xrightarrow{*}_{may} m$ and $m \xrightarrow{t}_{may} m'$.*

Similarly, N is must-sound *iff*

- *for all $c \in C(i)$ and $m \in \mathfrak{M}(N)$ such that $[(i, c)] \xrightarrow{*}_{may} m$, we have $m \xrightarrow{*}_{must} [(f, c_1)]$ for some $c_1 \in C(f)$, and*
- *for every $t \in T$ there exists a color $c \in C(i)$ and two markings $m, m' \in \mathfrak{M}(N)$ such that $[(i, c)] \xrightarrow{*}_{must} m$ and $m \xrightarrow{t}_{must} m'$.*

We now lift the notion of refinement to CWF-nets and show that it preserves/reflects soundness. As termination plays the central role in the definition of soundness, we extend Definition 2 with a requirement that final marking are only bisimilar to final markings.

Definition 5 (Termination-preserving refinement). *Let N_1 and N_2 be two CWF-nets. A refinement $R \subseteq \mathfrak{M}(N_1) \times \mathfrak{M}(N_2)$ is* termination-preserving *if, for every $(m_1, m_2) \in R$, $m_1 = [(f_1, c_1)]$ for some $c_1 \in C(f_1)$ iff $m_2 = [(f_2, c_2)]$ for some $c_2 \in C(f_2)$. We write $N_2 \preccurlyeq_{term} N_1$ if there is a termination-preserving refinement R such that $([(i_1, c_1)], [(i_2, c_2)]) \in R$ for every $c_1 \in C(i_1)$ and $c_2 \in C(i_2)$.*

Theorem 6. *Let N_1 and N_2 be two CWF-nets. If $N_2 \preccurlyeq_{term} N_1$, then*

- *if N_1 is must-sound, then so is N_2; and*
- *if N_2 is may-sound, then so is N_1.*

Proof. Let $N_2 \preccurlyeq_{\text{term}} N_1$ be witnessed by a refinement R such that $([(i_1, c_1)], [(i_2, c_2)]) \in R$. Suppose N_1 is must-sound and let $m_2 \in \mathfrak{M}(N_2)$ be such that $[(i_2, c_2)] \xrightarrow{*}_{\text{may}} m_2$. From the definition of refinement it follows (by a simple induction) that there is an $m_1 \in \mathfrak{M}(N_1)$ be such that $[(i_1, c_1)] \xrightarrow{*}_{\text{may}} m_1$ and

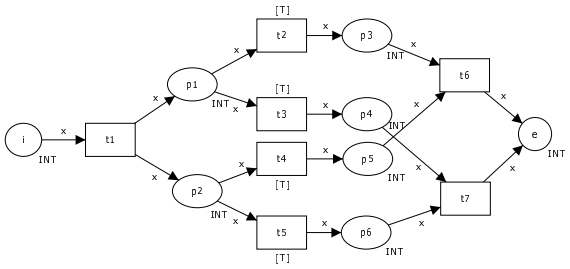

Fig. 3. A CWF-net that is may-sound but not must-sound

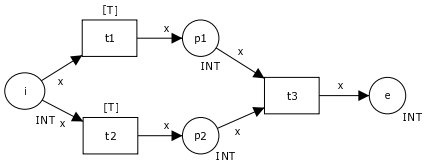

Fig. 4. A CWF-net that is not may-sound

$(m_1, m_2) \in R$. Since N_1 is must-sound, $m_1 \xrightarrow{*}_{\text{must}} [(f_1, c_1')]$. Again, by induction we have $m_2 \xrightarrow{*}_{\text{must}} m_2'$ and $([(f_1, c_1')], m_2') \in R$. Since R is termination-preserving, $m_2' = [(f_2, c_2')]$. As every must transition in N_1 must be simulated by a must transition in N_2, we also conclude that N_2 must have no dead transitions. From this we conclude that N_2 is must sound. The proof for the other case is analogue. □

We give two examples to illustrate the advantages of our approach compared to the currently used methods for checking workflow soundness. Consider the CWF-net in Fig. 3. This workflow is underspecified as the choice whether to take t_2 or t_3 (resp. t_4 or t_5) depends on the guard, which is $\top$. The standard soundness check on underdefined specifications ignores all data aspects and it would treat every guard as *true*. Therefore, a deadlock would be reported, e.g. in the marking $[p_3, p_6]$. Our approach, however, would report may-soundness (the workflow terminates if in the refinement both choices are made in the same way) but not must-soundness, resulting thus in the honest answer "I do not know, in some refinements it may be sound, and in others not". Indeed, if both the guard of t_2 and t_4, and the guard of t_3 and t_5, coincide, then the markings actually $[p_3, p_6]$ and $[p_4, p_5]$ are unreachable and the net is sound.

Consider now the simple CWF-net in Fig. 4. This workflow is reported unsound by the standard technique (due to deadlocks in $[p_1]$ and $[p_2]$). As the previous example suggests this still does not tell us anything about the behavior of this net in some refinement. However, our approach reports not may-soundness,

meaning that the workflow will be unsound in *any* possible refinement, directly implying that the deadlock error cannot be repaired by guard refinements only.

5 Data Refinement

The previous section introduced the notion of a refinement on the behavior of Coloured Petri nets. In this section we consider special types of structural refinements, called *data refinements*, and we prove them to be in agreement with behavioral ones. Figure 2 displays one example of a data refinement. Unlike structural refinements (e.g. place/transition refinements, subnet refinements [19, 12, 13]), the proposed refinement retains the structure of the net without modification but replaces data types (colours), guards and data computations by finer ones.

Definition 7 (Data refinement/abstraction). *Let* $N_1 = \langle P, T, \mathcal{A}, C_1, \mathcal{E}_1, G_1 \rangle$ *and* $N_2 = \langle P, T, \mathcal{A}, C_2, \mathcal{E}_2, G_2 \rangle$ *be two CPNs with identical sets of places, transitions and arcs. Let* $\alpha_p : C_2(p) \to C_1(p)$*, for* $p \in P$*, be some functions called the abstraction functions. A* data abstraction *is* $\alpha = \{\alpha_p \mid p \in P\}$*.*

Functions $\gamma_p : C_1(p) \to 2^{C_2(p)}$*, for* $p \in P$*, such that* $\gamma_p(a) = \{c \mid \alpha_p(c) = a\}$ *are called refinement functions. A* data refinement *is* $\gamma = \{\gamma_p \mid p \in P\}$*.*

We lift α for bindings by mapping the variable values to their abstract counterparts w.r.t. α. Intuitively, at the abstract level, we want to work with abstract data types and have "non-deterministic" operations on them. Consider for example the data type $Sign = \{neg, 0, pos, unknown\}$ (with intuitive meanings negative number, zero and positive number resp. and a possible refinement function mapping *pos* to $(0, +\infty)$, *neg* to $(-\infty, 0)$, 0 to $\{0\}$ and *unknown* to $(\infty; +\infty)$) where the + operation is defined as $pos + pos = pos; pos + 0 = pos; pos + neg = unknown$, etc. Another simple example is a client income abstraction for a bank business process: *Income-category* = {*high-income, middle-income, low-income*}. Data refinements allow refining them when there is a need.

We call a data refinement safe if it can only restrict the behaviour of the system by (possibly) turning some guards evaluated to $\top$ into *true* or *false*.

Definition 8 (Safe data refinement). *We say that* N_2 *is a* safe *data refinement of* N_1 *with respect to* γ*, denoted* $N_2 \trianglelefteq_\gamma N_1$*, if the following holds:*
$\forall p_1 \in P_1, t_1 \in T_1, m_1 \in \mathfrak{M}(N_1), b_1 \in \mathcal{B}(m_1, t_1) : b_2 \in \gamma(b_1) \Rightarrow$
$b(G(t_1)) = b_2(G_2(t_2)) \ \vee \ b(G(t_1)) = \top$.

Moreover, it is easy to see that the following holds.

Theorem 9. *If* $N_2 \trianglelefteq_{\gamma_1} N_1$ *and* $N_3 \trianglelefteq_{\gamma_2} N_2$*, then* $N_3 \trianglelefteq_{\{(\gamma_1 \circ \gamma_2)(p) \mid p \in P\}} N_1$*.*

Theorem 10. *Let* $N_2 \trianglelefteq_\gamma N_1$ *and* $m \in \mathfrak{M}(N_1)$*. Then for any* $m_2 \in \gamma(m)$ *we have* $(N_2, m_2) \preccurlyeq (N_1, m)$*.*

The way the present work on CWF-net may/must features preservation becomes close to the concept of safe abstraction within the *Abstract Interpretation* framework [5, 6, 22].

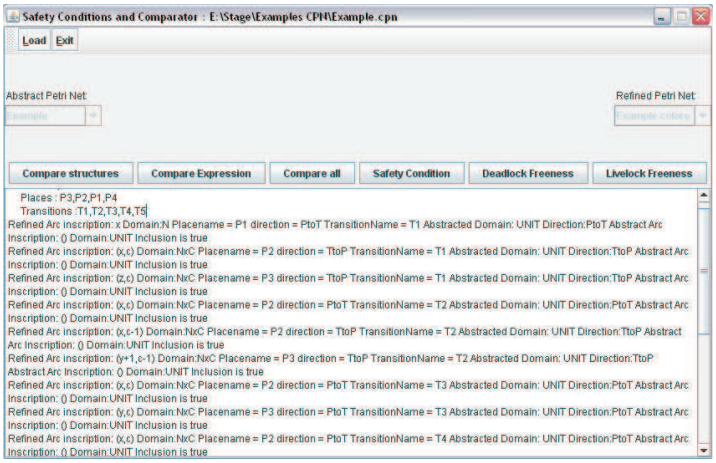

Fig. 5. Prototype sample run

6 Implementation

This section briefly reports on a prototype we have been developping to experiment with our refinement approach. This prototype, called Comparator, allows us to compare Petri nets structures through data refinements, to ensure must-deadlock freeness and must-livelock freeness under some conditions on data.

The prototype functioning is as follows. First of all, the examples of Petri nets are designed using CPNTools [17]. Using CPNTools allows storing all the information about a considered net in the `.cpn` file which is an XML file. Secondly, we require the *.cpn* files loaded into the prototype environment. Once the `.cpn` files of an abstract and a corresponding refined nets are loaded, our prototype parses them to extract usefull data. Thirdly, the prototype user verifies properties of interest. The algorithms implemented in Java are then executed, and meaningful and comprehensive messages about the property verification are displayed in the GUI, like in Fig. 5.

7 Conclusion

To cope with the complexity of concurrent systems, it is crucial to provide methods that enable debugging of a system specification in the early design phases. In this paper we introduced a framework for the verification of underdefined specifications based on Coloured Petri nets with the 3-valued logic for transition guards inducing may- and must-firings. We showed that our refinement notion is linked with data refinement and compatible with data refinement composition.

We formulated the requirements of may- and must-soundness for workflow nets and showed how they are preserved through refinements.

Future work. We plan to investigate refinements introduced in the present paper for particular classes of Petri nets, e.g. Free choice nets [9], for which we hope to obtain more efficient verification algorithms due to the net structure. We also intend to go further by investigating structural – e.g., place, transition, subnet, – refinements [19, 12, 13] w.r.t. may-/must-enabledness and soundness.

Related work. The use of abstractions/refinements in the verification is well-studied for many formalisms (see e.g. [18, 20]). Refinement design frameworks preserving (P)LTL formulas were studied for B refinement [7] and Z refinement [8]. Our approach allows the preservation of a broader class of properties. The refinement we introduce is in fact an adaptation of the termination-preserving refinement from [21] to the CWF-nets framework, implying that it preserves μ-calculus properties. When fixing a framework for operation types, the present work on refinement becomes closely related to the concept of safe abstractions, which is well-developed within the *Abstract Interpretation* framework [5, 6, 22]. The verification of incomplete state spaces as partial Kripke structures and a 3-valued interpretation to modal logic formulas on these structures were investigated in [4]. Our work uses similar ideas in a different context.

Acknowledgement. We thank C. Bassetti, Ch. Bon and M.N. Irfan for their help in implementing algorithms.

References

1. W. Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W. Aalst, L. Aldred, M. Dumas, and A. Hofstede. Design and Implementation of the YAWL System. In *Advanced Information Systems Engineering, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *LNCS*, pages 142–159. Springer, 2004.
3. W. M. P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, ICATPN'1997*, volume 1248 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
4. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In N. Halbwachs and D. Peled, editors, *11th Int. Conf. on Computer Aided Verification CAV '99*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 1999.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, New York, NY, USA, 1977. ACM Press.
6. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD dissertation, Eindhoven University of Technology, July 1996.
7. C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement preserves PLTL properties. In D. Bert, J. P. Bowen, S. King, and M. A. Waldén, editors, *ZB 2003: Formal Specification and Development in Z and B, Third Int. Conf. of B and Z Users*, volume 2651 of *Lecture Notes in Computer Science*, pages 408–420, 2003.
8. J. Derrick and G. Smith. Linear temporal logic and z refinement. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Algebraic Methodology and Software*

Technology, 10th International Conference, AMAST 2004, volume 3116 of *Lecture Notes in Computer Science*, pages 117–131, 2004.
9. J. Desel and J. Esparza. *Free Choice Petri nets.*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
10. Examples of Industrial Use of CP-nets. www.daimi.au.dk/CPnets/intro/example_indu.html.
11. N. Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986.
12. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In W. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003, ICATPN'2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 337–356. Springer-Verlag, 2003.
13. H. Huang, T.-Y. Cheung, and W. M. Mak. Structure and behaviour preservation by Petri-net-based refinements in system design. *Theoretical Computer Science*, 328(3):245–269, 2004.
14. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical*. Springer-Verlag, 1992.
15. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer, 1997.
16. K. Jensen, L. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.
17. K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 9(3-4):213–254, 2007.
18. Y. Kesten, Z. Manna, and A. Pnueli. Temporal verification of simulation and refinement. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, volume 803 of *Lecture Notes in Computer Science*, pages 273–346. Springer, 1994.
19. C. Lakos. Composing abstractions of coloured Petri nets. In *Application and Theory of Petri Nets 2000, ICATPN'2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 323–345. Springer-Verlag, 2000.
20. L. Lamport. The temporal logic of actions. *ACM Trans. On Programming Languages and Systems*, 16(3):872–923, 1994.
21. K. G. Larsen. Modal specifications. In *Int. Ws. on Automatic Verification Methods for Finite State Systems, 1989*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1990.
22. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
23. N. Trčka. Workflow Soundness and Data Abstraction: Some negative results and some open issues. In *In Workshop on Abstractions for Petri Nets and Other Models of Concurrency (APNOC)*, pages 19–25, 2009.