Lorenzo Capra and Walter Cazzola. Evolutionary Design through Reflective Petri Nets: an Application to Workflow. In Proceedings of the 26th IASTED International Conference on Software Engineering (SE'08), pages 200–207, Innsbruck, Austria, February 2008. ACTA Press.

EVOLUTIONARY DESIGN THROUGH REFLECTIVE PETRI NETS: AN APPLICATION TO WORKFLOW

Lorenzo Capra
Department of Informatics and Communication
Università degli Studi di Milano
Milan, Italy
email: capra@dico.unimi.it

Walter Cazzola
Department of Informatics and Communication
Università degli Studi di Milano
Milan, Italy
email: cazzola@dico.unimi.it

ABSTRACT
The design of dynamic workflows needs adequate modeling/specification formalisms and tools to soundly handle possible changes during workflow operation. A common approach is to pollute workflow design with details that do not regard the current behavior, but rather evolution. That hampers analysis, reuse and maintenance in general. We propose and discuss the adoption of a recent Petri net-based reflective model as a support to dynamic workflow design. Keeping separated functional aspects from evolution, results in a dynamic workflow model merging flexibility and ability of formally verifying basic workflow properties. A structural on-the-fly characterization of sound dynamic workflows is adopted based on Petri net's free-choiceness preservation. An application is presented to a localized open problem: how to determine what tasks should be redone and which ones do not when transferring a workflow instance from an old to a new template.

KEY WORDS
Dynamic Workflow, Free-Choice Petri nets, Reflection.

1 Introduction

Change occurs frequently in business processes due to two primary reasons [22]: i) at design time the workflow specification is incomplete due to lack of knowledge, ii) errors or exceptional situations can occur during the workflow execution; these are usually tackled on by deviating from the static schema, and may cause breakdowns, reduced quality of services, and inconsistencies.

Most of existing workflow management solutions (e.g., IBM Domino, iPlanet, Fujisu iFlow, TeamCenter) are designed to handle static business processes, in various degrees. The solution currently adopted by most WMS is in fact that, once process changes occur, new workflow templates are defined and workflow instances are initiated accordingly from scratch. This over-simplified approach forces tasks that were completed on the old instance to be executed again, also when not necessary. If the workflow is complex and/or involves a lot of external collaborators, a substantial business cost will be incurred.

Dynamic workflow change management might be brought in as a potential solution. Formal techniques and analysis tools can support the development of adaptable WMS capable to handle undesired results introduced by workflow dynamism.

In the research field on dynamic workflow, the prevalent opinion is that models that capture work practices should be based on a formal theory and be as simple as possible [2]. The Milano system [1] aims to provide process models (i.e., workflow templates) as 'resources for action' rather than strict blueprints of work practices. May be the most famous dynamic workflow formalization, the ADEPTflex system [17], is designed to support dynamic change at runtime. A complete and minimal set of change operations is defined. The correctness properties defined by ADEPTflex are used to determine whether a specific change can be applied to a given workflow instance or not.

Most of workflow modeling techniques are based on Petri Nets (PN) [19], due to PN's description efficacy, formal nature, and the availability of consolidated PN-based analysis techniques. Classical PNs have a fixed topology, so they are well suited to model workflow matching a static paradigm. Conversely, the design of dynamic workflows is not well supported by classical PNs, dynamism/evolution must be hard-wired in the PN model and bypassed when not in use. That requires some expertise in PN modeling, and might result in an incorrect or partial description of workflow behavior. What is even worst, analysis would be polluted by a great deal of details concerning evolution.

Separating evolution from the (current) system behavior is worthwhile. This concept has been recently applied to a PN-based context [5], using reflection [15] as mechanisms that easily permits separation of concerns. A basic reflective model layered in two causally connected levels (base-, and meta-level) is used.

With respect to several 'dynamic' PN extensions recently appeared (e.g. [4, 12], as concerns specifically the workflow field [3, 9, 10, 14]) reflective Petri nets [5] do not define a new PN paradigm, but rely upon classical PNs. That gives the possibility of using available tools and consolidated analysis techniques in a fully orthogonal fashion.

We propose and discuss the adoption of reflective PN as a support to dynamic workflow design, considering a localized open problem: how to determine what tasks should be redone and which ones do not when transferring a workflow instance from an old to a new template. The prob-

lem is efficiently, but rather empirically, addressed in [16], where a template-based schema is implemented, relying on the concept of *bypassable* task. Conforming to the same concept, we propose an alternative, parametric solution, that allows evolutionary steps to be soundly formalized and basic workflow properties to be verified. In particular, a structural on-the-fly characterization of sound dynamic workflows is adopted.

According to [1,2], the current (base) workflow model is kept as simple as possible. Our approach has some resemblance also with [17], sharing the same completeness/minimality criteria, but considerably differs in management of changes: it neither provides exception handling nor undoing mechanism of temporary changes, rather it relies upon a sort of on-the-fly check of properties.

The paper is structured as follows: in section 2 we introduce a few basic notions related to usage of PNs for workflow modeling; in section 3 we outline the PN-based reflective model, introducing the adopted terminology and the language used to specify the evolutionary strategy; in section 4 we recall the (template-based) approach to dynamic workflow change defined in [16], then we present and discuss an alternative based on reflective PN that exploits PN structural analysis to ensure a sound evolution of processes; finally in section 5 we draw our conclusions and perspectives.

2 PN and WF: background

This section introduces terminology and notations of the base-level Petri net class we shall use in the sequel. Basic concepts and properties related to the use of PN in workflow modeling are also given. We refer to [18, 20] for more elaborate introductions.

Definition 1 *(Petri net) A Petri net is a triple* $(P; T; F)$*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \emptyset$*;),*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

Symbols ${}^\bullet n, n^\bullet$ denote the pre/post sets of a node $n \in P \cup T$, respectively. The extensions ${}^\bullet A, A^\bullet$, $A \subseteq P \cup T$ will be also used. The neighbor set of n, ${}^\bullet n \cup n^\bullet$, is denoted n°. Note that in the context of workflow procedures it makes no sense to have weighted arcs, because places correspond to conditions. A marking (state) M is a distribution of tokens over places, i.e., $M \in Bag(P)$.

Transitions change the state of the net according to the following firing rule:
- A transition t is said to be enabled in M iff each place $p \in {}^\bullet t$ contains at least one token.
- An enabled transition t may fire. If t fires, then it consumes one token from each $p \in {}^\bullet t$ and produces one token for each $p \in t^\bullet$.

Given a Petri net $(P; T; F)$ and a state M_1, we have the following notations:

$M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ leads from M_1 to M_n via $M_2, \dots, M_{n-1}$
M_n is *reachable* from M_1 iff there exists σ, $M_1 \xrightarrow{\sigma} M_n$.
$(PN; M)$ denotes a Petri net with an initial state M. Given $(PN; M)$, M' is said reachable iff it is reachable from M.

Let us define a few standard properties for Petri nets. First, we define properties related to the dynamics of a Petri net, then we give some structural properties.
(Live). $(PN; M)$ is live iff, for every reachable state M' and every transition t there exists M'' reachable from M' which enables t.
(Bounded, safe). $(PN; M)$ is bounded iff for each place p there exists $n \in \mathbb{N}$ such that for every reachable state the number of tokens in p is less than n. A bounded net is safe iff for each place $n \leq 1$. A marking M of a safe net is an element of the power-set on P.
(Path). A path C from a node n_1 to a node n_k of a PN is a sequence $n_1, n_2, \dots, n_k$ such that $(n_i, n_{i+1}) \in F$ for $1 \leq i \leq k-1$.
(Free-choice). A Petri net is free-choice iff, for every pair of transitions t_1 and t_2, ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset \Rightarrow {}^\bullet t_1 = {}^\bullet t_2$.
(Causal connection - CC). transition t_1 is causally connected to t_2 iff $(t_1{}^\bullet \setminus {}^\bullet t_1) \cap {}^\bullet t_2 \neq \emptyset$.
CC^* denotes the unreflexive transitive closure of CC.

2.1 WF-nets and Free-Choiceness

A Petri net can be used to specify the control flow of a WF. Tasks are modeled by transitions and causal dependencies by places and arcs. A place corresponds to a task pre- and/or post-condition. A Petri net which models a workflow is called WF-net.

Definition 2 *(WF-net). A Petri net PN =* $(P; T; F)$ *is a WF-net iff:*

❶ *There is one source place* i *such that* ${}^\bullet i = \emptyset$.

❷ *There is one sink place* o *such that* $o^\bullet = \emptyset$.

❸ *Every node* $x \in P \cup T$ *is on a path from* i *to* o.

A WF-net has exactly one input place (i) and one output place (o), because any case handled by the procedure represented by the WF-net is created when it enters the WMS and is deleted once it is completely handled by the WMS, i.e., the WF-net specifies the life-cycle of a case. The third requirement in definition 2 avoids dangling tasks and/or conditions, i.e., tasks and conditions which do not contribute to the processing of cases.

If we add to a WF-net PN a transition t^* such that ${}^\bullet t^* = o$ and $t^{*\bullet} = i$, then the resulting Petri net $\overline{PN}$ (called the *short-circuited* net of PN) is strongly connected.

The requirements stated in definition 2 only relate to the structure of the Petri net. However, there is another requirement that should be satisfied:

Definition 3 *(soundness) A procedure modeled by a WF-net PN =* $(P; T; F)$ *is sound iff:*

❶ *For every state M reachable from state $\{i\}$, there exists σ, $M \xrightarrow{\sigma} \{o\}$*

❷ *$\{o\}$ is the only state reachable from $\{i\}$ with at least one token in place o.*

❸ *There are no dead transitions in $(PN;\{i\})$: $\forall t \in T$, there exists M reachable from $\{i\}$, $M \xrightarrow{t} M'$*

In other words, for any case the procedure will terminate eventually (in the context of workflow management we reasonably assume a strong notion of fairness) and the moment the procedure terminates there is a token in place o and all the other places are empty. That is sometimes referred to as *proper termination*. Moreover, it should be possible to execute an arbitrary task by following the appropriate route though the WF-net.

The soundness property relates to the dynamics of a WF-net, and may be considered as a basic requirement for a workflow. Standard Petri-net-based analysis techniques can be used to verify soundness. A WF-net PN in fact is sound if and only if the short-circuited net $(\overline{PN};\{i\})$ is *live and bounded* [20]. Despite that useful characterization of the quality of a workflow process, for arbitrary WF-nets it may be intractable to decide soundness: liveness and boundedness are decidable, but also EXPSPACE-hard.

Therefore structural characterizations of sound WF-nets were investigated [20]. Free-Choice (FC) Petri nets seem to be a good compromise between expressive power and 'analyzability', and are the largest class of Petri nets for which strong theoretical results and efficient analysis techniques exist [8]. In particular, as shown in [20], soundness of a FC WF-net (as well as many other problems) can be decided in *polynomial* time. Moreover, a sound FC WF-net $(PN;\{i\})$ is guaranteed to be *safe*, according to the interpretation of WF-net places as conditions.

Another good reason for restricting WF-nets to FC Petri nets is that the routing of a case should be independent of the order in which tasks are executed. If non-FC Petri nets are allowed, then the choice between conflicting tasks may be influenced by the order in which the preceding tasks are executed. In literature the term confusion is often used to refer to a situation where the FC property is violated by a badly mixture of parallelism and choice.

The FC property is a desirable property for workflow procedure. If a workflow can be modeled as a FC WF-net, one should do so. Most of existing WMSs only allow for FC workflows. In our application of reflective PN model we will admit as base-level Petri nets only FC WF-nets.

Although FC WF-nets are a satisfactory characterization of workflow well-structuredness, there are non-FC WF-nets which correspond to sensible workflows. S-coverability [20] is a (structural) generalization of the FC property: a sound FC WF-net is S-coverable. Unfortunately, in general, it is not possible to verify soundness of an arbitrary S-coverable WF-net in polynomial time, this problem being in fact PSPACEcomplete.

3 Reflective PNs

The *reflective Petri net* approach [5] permits developers to model a (discrete-event) system and *separately* all its possible evolutions, and to dynamically adapt system's model when evolution must occur.

The approach is based on a reflective architecture [6] structured into two logical layers. The first layer, called *base-level*, is represented by the PN modeling the system prone to be evolved, also called *base-level PN*; whereas the second layer, called *meta-level* is represented by the *meta-program*, following the reflection parlance, a Colored PN [13] composed by the *evolutionary strategies* that will drive the evolution of the base-level PN when certain events occur. Entities on the meta-level perform computations on entities residing on the lower level.

The *reflective framework*, realized by a CPN as well, is responsible for really carrying out the evolution of the base-level PN at the meta-level. Meta-level computations in fact operate on a representative of the lower-level, called *reification*. The base-level PN reification is defined as a *marking* of the reflective framework, and is updated every time the base level Petri net enters a new state. The reification is used by the meta-program to observe (*introspection*) and manipulate (*intercession*) the base-level PN. Each change to the reification is reflected on the base-level PN at the end of a meta-computation (*shift-down action*), i.e., the base-level PN and its reification are *causally connected*, the reflective framework being responsible for that.

According to the reflective paradigm, the base-level PN runs irrespective of the meta-program, being not aware of the existence of a meta-level. The meta-program is implicitly activated (*shift-up action*), and a suitable strategy is then put into action, under two conditions: i) either when the base-level PN model reaches a given configuration, or ii) when triggered by an external/unpredictable event.

Intercession on the base-level PN is carried out in terms of a minimal set of basic operations (called the *evolutionary interface*), that permit any kind of base-level's evolution, both at structure (topology) and marking (current state) level: the meta-programmer can add/remove places, transitions and arcs, and freely move tokens all over the base-level PN places. The evolutionary strategy specifies arbitrarily complex transformation patterns for the base-level Petri net. To simplify their design we have provided the developer with a tiny, ad-hoc (meta-)language that allows everyone to specify his own strategy in a simple and formal way. The syntax is inspired by Hoare's CSP [11], enriched with a few specific constructs and notations for easy manipulation of nets. A strategy specified in this way can be automatically translated into a corresponding CPN, that will be in turn composed to the evolutionary framework to obtain the whole meta-model.

Evolutionary strategies have a *transactional* semantics: either they succeed, or leave the base-level PN unchanged. Several strategies could be candidate for execution at a given instant: the simplest policy (here adopted)

selects one non-deterministically.

The interaction between base-level and meta-level, and between meta-level entities, has been formalized in [5]. Let us only outline the essential aspects:

— the structure of the reflective framework is fixed, while the evolutionary strategies are coupled to the base-level PN, and change from time to time;

— the reflective framework and the meta-program are separated components, sharing two disjoint sets of boundary places: the base-level PN reification and the evolutionary interface; the interaction between components is realized through *place superposition*;

— the base-level PN reification is observed and manipulated by the meta-program; whereas the evolutionary interface allows the evolutionary strategies to send evolutionary commands to the reflective framework (that operate them);

— the base-level reification color domains are similar to formal parameters, they are bound from time to time to a given base-level PN; reification's initial marking corresponds to the initial base-level configuration.

The whole reflective architecture is characterized by a fixed part (the reflective framework), and by a part varying from time to time (the base-level PN and the evolutionary strategy). The fixed part is used to put evolution into practice for any kind of system. It is responsible for the reflective behavior of the architecture, hiding the work of the evolutionary sub-system to the base-level PN. This approach permits a clean separation between the PN describing the evolution and the model of the evolving system, that is updated only when necessary. So the base-level PN model is not polluted by details related to evolution.

4 Template-Based Workflow Evolution

An interesting solution to facilitate efficient dynamic workflow change is proposed in [16]. The approach addresses template-based dynamic workflow change, according to a consolidated industrial practice, and is implemented in SmarTeam, a leading PDM system. The idea is to identify all nodes in the new workflow instance that satisfy the following conditions i) they are unchanged, ii) they have finished in the old workflow instance, and iii) they need not be executed again, i.e., they are bypassable.

Two nodes (transitions in PN parlance) are identical, before and after change, iff they represent the same tasks and preserve input/output connections. It is hereafter assumed that two nodes represent the same tasks iff they are name preserving. The output of a node is affected by all nodes from which there is a path to the node itself. Therefore, to determine if a node/task that was completed in the old instance is bypassable when the instance is transferred to a new template, an additional condition is needed: all nodes from which there is a path to the node itself, must be bypassable. This solution has been implemented in Dassault's SmarTeam PDM system.

4.1 A Reflective PN Approach

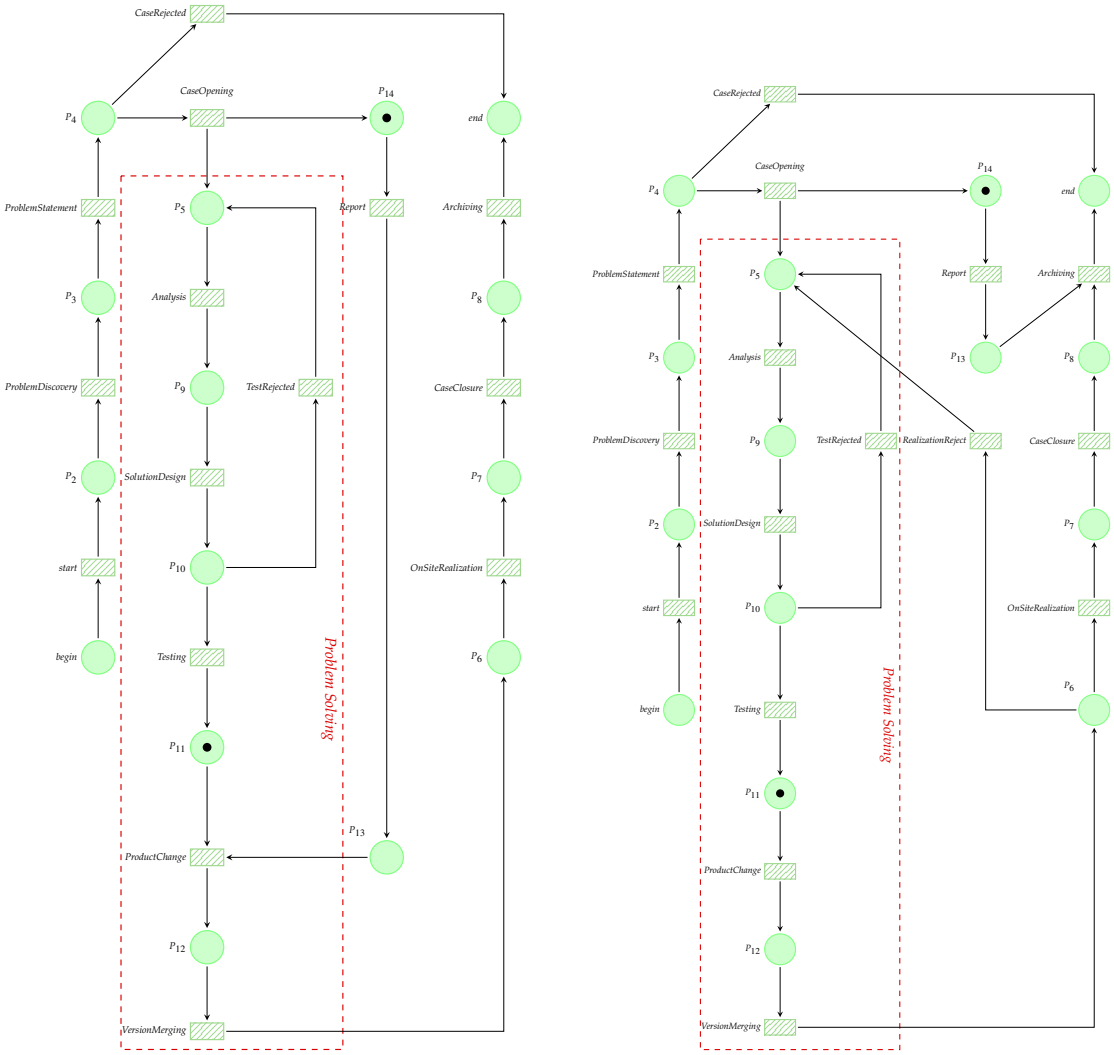
Our alternative to [16]'s approach based on reflective PNs allows a sounder formalization of evolutionary steps, and permits some kind of on-the-fly validation of workflow change by means of a simple structural analysis performed on the WF-net reification. In particular, free-choiceness preservation is checked. Changes are not reflected to the base-level WF-net in case of a negative check. The aim is to show that reflective Petri nets can be helpful in designing dependable dynamic workflows.

We consider the same case presented in [16] (Figure 1). A company has several regional branches. To enhance operation consistence, the company headquarter (HQ) standardizes its business processes in all branches. A workflow template is defined to handle customer problems. When the staff in a branch encounters a problem, a workflow instance is initiated from the template and executed until completion. The PN specification of the template is given in Figure 1(a). In the template, a problem goes through two stages: problem solving and on-site realization. Problem solving involves several steps, included in the dashed box in Figure 1(a)). When opening a case, the staff in the branch reports the case to the HQ. When closing the case, the staff archives the related documents. The HQ manages all instances related to the problem handling process. In response to business needs, the HQ may decide to change the problem handling process and to transfer old workflow instances to the new template (Figure 1(b)), that differs from the original one basically in two points: a) "reporting" and "problem solving" are completely separated tasks; b) "on site realization" task can fail, in that case the "problem solving" sequence restarts.

When using reflective PN, the evolutionary schema has to be completely reviewed. The new workflow template is not passed as input to the staff of the company branches, but it results from applying an evolutionary strategy to a workflow instance belonging to the current template. The initial base-level PN is assumed to be a *free-choice WF-net*.

No details related to the workflow dynamics are hardwired in the base-level net. Evolution is delegated to the meta-program, that acts on the WF-net reification. The meta-program is activated when an evolutionary signal is received from HQ, or some anomaly (e.g., a deadlock) is revealed by introspection. Introspection is also used to discriminate whether the evolutionary commands can be safely applied to the current workflow instance, or they have to be temporarily discarded.

Figure 1 depicts the following situation: a workflow instance running on the old (base) template (figure 1(a)) receives a message from HQ. The current marking represents a state where the "solution design" sub-task of problem solving and the "report" task are pending, and a number of tasks (e.g., "analysis" and "case opening") have fin-



(a) The Petri net modeling the base workflow template

(b) A possible workflow evolution

Figure 1. A basic template and its possible evolution.

ished. The meta-program in that case successfully operates change on the old template instance, once verified that all pending tasks have been enabled by sequences of bypassable tasks only. Evolutionary commands are sent as output commands by HQ. The workflow instance running on the modified template is illustrated in figure 1(b)). One might think that this approach is instance-based, rather than template-based. In truth it covers both schema: if the evolutionary commands are broadcasted to all workflow instances we fall in the second category.

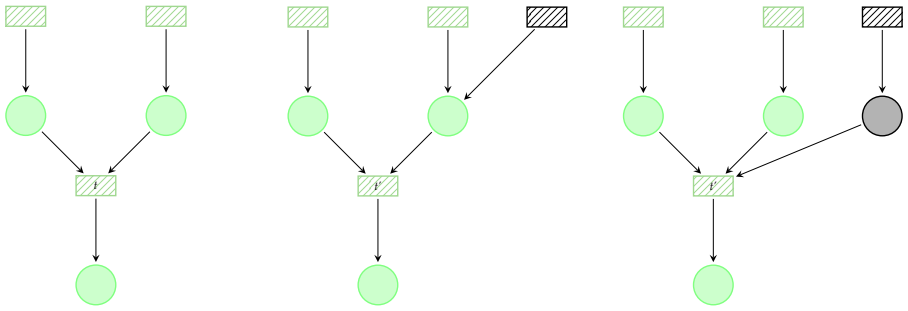
The evolutionary strategy relies upon the definition of *adjacency preserving* node, more general than the *unchanged* node notion used in [16]. It is inspired by van der Aalst's general concept that a dynamic workflow change must preserve the inheritance relationship between old and new workflow templates [21].

Let us first introduce some *notations*. Symbols x,$\overline{x}$ will be used to denote the same node before and after change, respectively: x belongs to a WF-net PN, $\overline{x}$ belongs to the net PN' resulting from change. NEW_P, DEL_P, NEW_T, DEL_T, and NEW_A, DEL_A, denote the base level places/transitions/arcs to be added and removed, respectively; $DEL\_N = DEL\_P \cup DEL\_T$, $NEW\_N = NEW\_P \cup NEW\_T$. The workflow reification nodes/arcs before change are denoted by OLD_N, OLD_A. Finally, NO_ADJ, NO_BYPS denote the set of nodes not preserving adjacency and non-bypassable, respectively ($NO\_ADJ \subseteq NO\_BYPS$).

Definition 4 *(adjacency preserving transition)* *Let* $\mathcal{A}_t = {}^\bullet(t^\circ) \cup (t^\circ)^\bullet$ *(the set of transitions adjacent to* t*).* t *is adjacency preserving iff* $\mathcal{A}_{\overline{t}} \cap OLD\_N = \mathcal{A}_t$ *and there exist a bijection* $\varphi : t^\circ \to \overline{t}^\circ$ *such that* $\forall x \in \mathcal{A}_t \forall y \in t^\circ$,

$$y \in {}^\bullet x \text{ iff } \varphi(y) \in {}^\bullet\overline{x} \text{ and } y \in x^\bullet \text{ iff } \varphi(y) \in \overline{x}^\bullet$$

In practice a task t is adjacency preserving iff all its causal-



(a) initial situation (b) change adjacency preserving (c) change not preserving adjacency

Figure 2. Definition 4 illustrated.

ity/conflict relationships to adjacent tasks are maintained by change. A case where Definition 4 holds, and another where it does not, are illustrated in figure 2, where the black box denotes a new task. Checking definition 4 is computationally expensive. However, if some reasonable assumptions of change well-definiteness are made (e.g., if useless changes such as "deleting a given place p, then adding p' inheriting p's connections", or "adding an arc $< p, t >$, then deleting p or t", are forbidden), check's complexity is greatly reduced. Lemma 1 states a set of rules for identifying a *superset* of nodes not preserving adjacency, that may be easily translated to an efficient meta-program subroutine. In most practical cases this superset coincides to the exact set.

Lemma 1 *Consider set N_a, built as follows*
$p \in DEL\_P \Rightarrow {}^\bullet p \cup p^\bullet \subseteq N_a$
$t \in DEL\_T \Rightarrow {}^\bullet({}^\bullet t) \cup (t^\bullet)^\bullet \subseteq N_a$
$< p, t > \in DEL\_A \vee < t, p > \in DEL\_A \Rightarrow {}^\bullet p \cup p^\bullet \subseteq N_a$
$< p, t > \in NEW\_A \wedge t \in OLD\_N \Rightarrow \{t\} \cup A \subseteq N_a$,
where $A = {}^\bullet p \cup p^\bullet$ *if* $p \in OLD\_N$, *else* $A = \emptyset$.
Then $NO\_ADJ \subseteq N_a$

The evolutionary meta-program corresponds to the CSP-like code in listing 1. The meta-program is activated at any transition of state on the current workflow instance (shift-up), reacting to three different types of events. In the case of deadlock, a signal is sent to HQ, represented by a CSP process identifier. If the current instance has finished, and a "new instance" message is received, the workflow is activated. Instead if there is an incoming evolutionary message from HQ, the evolutionary strategy starts running.

Just after an evolutionary signal, HQ communicates the workflow nodes/connections to be removed/added. For the sake of simplicity we assume that change can only involve workflow topology. The (super)set of non-bypassable nodes is then computed.

After operating the evolutionary commands on the current workflow reification, definition 2 and free-choiceness are checked out on the newly changed reification. Following, the strategy checks by reification introspection whether the suggested workflow change might cause a deadlock, or there might be any non-bypassable tasks causally-connected to an old task which is currently pending. In either case, a restart procedure takes the workflow reification back to the state before strategy's activation. Otherwise, change is reflected to the base-level (shift-down). To avoid otherwise possible inconsistencies, the base-level is "frozen" during strategy's execution [5]. The proposed schema might be adopted as a template for a class of arbitrarily complex evolutionary patterns.

Language built-ins and routine calls are in bold. The $NODE$ type represents a (logically unbounded) recipient of base-level nodes, and is partitioned into $PLACE$ and $TRAN$ subtypes . A particular version of CSP repetitive command is used: letting set E be *finite*, `*(e in E)[ «command» ]` makes `command` to be executed iteratively for each $e \in E$. Note the overloading of operator '*', which is used also to denote the set intersection. The `exists` quantifier is used to check whether a net element is currently reified.

The built-in routine **ReifNodes** computes the nodes belonging to the current base-level reification. The routine **notAdjPres** initializes the set of non-bypassable nodes, according to lemma 1. The routines **ccTo** and **ccBy** compute the set of nodes that routine's argument is causally connected to, and that are causally connected to routine's argument, respectively. Listing 2 expands the routine checking preservation of base evel free-choiceness (**checkFc**).

Let us explain how the strategy works considering again Figure 1. After receiving evolutionary commands:
-NEW_PLACE={};DEL_NODE={}
-NEW_TRAN={RealizationRejected};
-DEL_ARC={$\langle p13, \texttt{ProductChange}\rangle$};
-NEW_ARC={$\langle p6, \texttt{RealizationRejected}\rangle$,
$\langle p13, \texttt{Archiving}\rangle, \langle \texttt{RealizationRejected}, p5\rangle$}.

The non-bypassable tasks come to be: `Report`, `Archiving`, `ProductChange`, `OnSiteRealization`, `CaseClosure`. In the new workflow instance tasks

```
*[
   VAR p, t, n : NODE;
   VAR NEW_P,NEW_T,OLD_N={},
       DEL_N, NO_BYPS : SET(NODE);
   VAR NEW_A,DEL_A: SET(ARC);

   HQ ? change-msg() → [
   //receiving evolutionary commands
    HQ ? NEW_P; HQ ? NEW_T; HQ ? NEW_A;
    HQ ? DEL_A; HQ ? DEL_N;
   //computing workflow reification
    OLD_N = ReifiedNodes();
   //computing nonbypassable tasks
    NO_BYPS = ccTo(notAdjPres());
   //changing reification
    newNode(NEW_P+NEW_T); newArc(NEW_A);
    deleteArc(DEL_A); delNode(DEL_N);
    //checking well-formedness
    checkWfNet(); checkFc();
    /*if there might be a deadlock,
     or a nonbypassable task CC to a
     pending one ...*/
    !exists t in Tran, enab(t) or
     (exists t in Tran * OLD_N, enab(t)
       and !isEmpty(ccBy(t)* NO_BYPS))
        → [ restart()] //no change
    shiftDown() ] //change reflected
  □
    #end=0 and
    !exists t in Tran, enab(t) →
      [ HQ ! notify-deadlock()]
  □
    #end=1; HQ ? newInstance-msg() →
      [flush(end); incMark(begin)]
]
```

Listing 1. workflow evolutionary strategy

```
[ VAR t,p,t1 : NODE;
  VAR CHECKED={} : SET(NODE);

  *(<p,t> in NEW_A + DEL_A)
    [ exists(p) and exists(t) → [
      *(t1 in post(p) \ CHECKED)   [
       t1 <> t and pre(t) <> pre(t1)
            → [ restart()]
       CHECKED = CHECKED + post(p) ]]
 ]]
```

Listing 2. piece of code checking free-choiceness

`Report` and `ProductChange` are pending (enabled) in the current marking $M : \{p_{11}, p_{14}\}$ of the net in figure 1(b). All old completed tasks that are causally connected to one of them can be bypassed, so the new workflow has not to be restarted from scratch, saving a lot of work.

As a counter example assume that the only pending task is `OnSiteRealization` (the current state of the net in figure 1(a) would be $M' : \{p_6\}$), meaning that, among other, tasks `ProductChange`, `VersionMerging` and `Report` have finished in the old workflow instance: change in that case is delayed till the instance completion.

If the suggested change were carried out (reflected) without any consistency control, a deadlock would be eventually entered (state $\{p_8\}$) after the process continues on the modified template.

The mechanism just described ensures a dependable evolution of workflow instances, while being enough flexible. Our aim was not to propose a general solution to the problem addressed in [16]. Better, sounder and more effective policies do probably exist. Rather, we aimed at showing how an approach merging consolidated reflection concepts to classical PN techniques can be suitably adopted to cope with the critical issues of dynamic workflow change. In particular, basic properties of workflows can be checked on-the-fly, discarding suggested changes when necessary.

The base-level PN, which is guaranteed to be a free-choice WF-net during its evolution, may be analyzed using different, sound polynomial techniques. Structural techniques, in particular, are elegant and very efficient, but in general they are highly affected by model complexity. Keeping evolutionary aspects separated from functional aspects encourages their usage.

By operating the structural algorithms of GreatSPN tool [7], it is possible to discover that both models in Figure 1 are covered by *place-invariants*. Thereby a lot of interesting properties descend: in particular boundedness and liveness, i.e., workflow soundness.

4.2 A counter example

Assume that evolution takes place when the only pending task is `OnSiteRealization` (current state of the net in figure 1(a) $M' : \{p_6\}$), that means, among other, tasks `ProductChange`, `VersionMerging` and `Report` have finished: change in that case is temporarily discarded by the evolutionary strategy, after verifying that some nonbypassable tasks are causally connected to the pending one.

If the suggested change were carried out (reflected) without any consistency control, a deadlock would be eventually entered (state $\{p_8\}$) after the process continues running on the modified template. The problem is that M' is not a reachable marking of $(PN'; \{begin\})$, but reachability is NP-complete in live and safe free-choice Petri nets, so it would make no sense to check reachability at meta-program level.

5 Conclusion

Covering the intrinsic dynamism of modern processes has been widely recognized as a challenge by designers of workflow management systems. PN are a central model of workflows, but traditionally they have a fixed structure. We have proposed and discussed the adoption of reflective PN as a formal model for designing sound, template-based dynamic workflows. A clean separation between the current behavior and the evolution of a workflow, and the use of efficient PN structural techniques, make it possible to check basic workflow properties while evolution is in progress. As an application, an algorithm is delivered to soundly transferring workflow instances from an old to a new template. Ongoing research is in two directions: i) integrating the approach into the GreatSPN package, ii) using a high-level PN class also for the base-level of the reflective model, to incorporate both resources and data in the process description.

References

[1] A. Agostini and G. De Michelis. Modeling the Document Flow within a Cooperative Process as a Resource for Action. Technical report, CTL-DSI, University of Milano, 1996.

[2] A. Agostini and G. De Michelis. A Light Workflow Management System Using Simple Process Models. *Computer Supported Cooperative Work (CSCW'00)*, 9(3-4):335–363, Aug. 2000.

[3] E. Badouel and J. Oliver. Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems. IRISA Research Report PI-1163, IRISA, Jan. 1998.

[4] L. Cabac, M. Duvignau, D. Moldt, and H. Rölke. Modeling Dynamic Architectures Using Nets-Within-Nets. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, LNCS 3536, pages 148–167, Miami, FL, USA, June 2005. Springer.

[5] L. Capra and W. Cazzola. A Petri-Net Based Reflective Framework. In F. Arbab and M. Sirjani, editors, *Proceedings of the IPM International Workshop on Foundations of Software Engineering (FSEN'05)*, Electronic Notes in Theoretical Computer Science 159, pages 41–59, Tehran, Iran, on 1st-3rd of Oct. 2005. Elsevier.

[6] W. Cazzola. Evaluation of Object-Oriented Reflective Models. In *Proceedings of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (EWROOPS'98)*, in 12th European Conference on Object-Oriented Programming (ECOOP'98), Brussels, Belgium, on 20th-24th July 1998. Extended Abstract also published on ECOOP'98 Workshop Readers, S. Demeyer and J. Bosch editors, LNCS 1543, ISBN 3-540-65460-7 pages 386-387.

[7] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, 24(1-2):47–68, Nov. 1995.

[8] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, New York, NY, USA, 1995.

[9] C. Ellis and K. Keddara. ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems. *Computer Supported Cooperative Work*, 9(3-4):293–333, Aug. 2000.

[10] A. Hicheur, K. Barkaoui, and N. Boudiaf. Modeling Workflows with Recursive ECATNets. In *Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNACS'06)*, pages 389–398, Timişoara, Romania, Sept. 2006. IEEE Computer Society.

[11] C. A. R. Hoare. *Communicating Sequential Processes*. Prantice Hall, 1985.

[12] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, LNCS 3536, pages 268–288, Miami, FL, USA, June 2005. Springer.

[13] K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets: Theory and Applications*. Springer-Verlag, 1991.

[14] M. Llorens and J. Oliver. Marked-Controlled Reconfigurable Workflow Nets. In *Prooceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNACS'06)*, pages 407–413, Timişoara, Romania, Sept. 2006. IEEE Computer Society.

[15] P. Maes. Concepts and Experiments in Computational Reflection. In N. K. Meyrowitz, editor, *Proceedings of the 2nd Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, volume 22 of *Sigplan Notices*, pages 147–156, Orlando, Florida, USA, Oct. 1987. ACM.

[16] Z.-M. Qiu and Y. S. Wong. Dynamic Workflow Change in PDM Systems. *Computers in Industry*, 58(5):453–463, June 2007.

[17] M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes in Workflow Management Systems without Losing Control. *Journal of Intelligent Information Systems*, 10((I2)):93–129, 1998.

[18] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[19] K. Salimifard and M. B. Wright. Petri Net-Based Modeling of Workflow Systems: An Overview. *European Journal of Operational Research*, 134(3):664–676, Nov. 2001.

[20] W. M. P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, The Netherlands, 1996.

[21] W. M. P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, Jan. 2002.

[22] W. M. P. van der Aalst and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, Sept. 2000.