Lorenzo Capra and Walter Cazzola. A Reflective PN-based Approach to Dynamic Workflow Change. In Proceedings of the 9th International Symposium in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'07), pages 533–540, Timişoara, Romania, September 2007. IEEE.

A Reflective PN-based Approach to Dynamic Workflow Change

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

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

Abstract

The design of dynamic workflows needs adequate modeling/specification formalisms and tools to soundly handle possible changes occurring during workflow operation. A common approach is to pollute design with details that do not regard the current workflow behavior, but rather its evolution. That hampers analysis, reuse and maintenance in general.

We propose and discuss the adoption of a recent Petri Net based reflective model (based on classical PN) as a support to dynamic workflow design, by addressing a localized 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.

Behind there is the idea that keeping functional aspects separated from evolutionary ones, and applying evolution to the (current) workflow template only when necessary, results in a simple reference model on which the ability of formally verifying typical workflow properties is preserved, thus favoring a dependable adaptability.

Keywords: Dynamic Workflow, Petri Nets, Reflection, Evolution.

1. Introduction

A dynamic workflow can change/evolve during execution. Change occurs frequently in current business processes due to two primary reasons [20]: 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 [16], is designed to support dynamic change at runtime. A complete and minimal set of change operations is defined that support users in modifying the structure of a running workflow. 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 workflow modeling techniques are based on Petri Nets (PN) [17], due to their description efficacy, formal nature, and ability to support correctness verification through consolidated techniques. Classical PNs have a fixed topol-

ogy, so they are well suited to model workflow matching a static paradigm, i.e, processes that are finished or aborted once they are initiated. 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. Even worst, analysis would be polluted by a great deal of details that concern evolution.

Separating evolution from the (current) system behavior is worthwhile. This concept has been recently applied to a PN-based context [5], using reflection [14] 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, 11] and, as concerns specifically application to the workflow field, [3, 8, 9, 13]) our model, called reflective Petri nets [5], does not define a new PN paradigm, but relies 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 the reflective PN-based model 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 problem is efficiently, but rather empirically, addressed in [15], where a template-based dynamic 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 workflow properties to be verified during evolutionary strategy execution.

According to [1, 2], the idea is to keep the current (base) workflow model as simple as possible. Our approach has some resemblance also with [16], sharing the same completeness/minimality criteria, but considerably differs in management of changes: it provides neither exception handling nor undoing mechanism of temporary changes at instance level, rather it relies upon a sort of on-the-fly checking of properties, through the 'reification' concept typical of reflection.

The paper is structured as follows: in section 2 we outline the PN-based reflective model, introducing the adopted terminology and the language used to specify the evolutionary strategy; in section 3 we introduce the dynamic workflow problem addressed in [15]; in section 4 we present a solution based on the reflective PN model; finally in section 5 we draw our conclusions and perspectives. We hereafter assume that the reader has some basic knowledge of classical PNs (specifically, safe `Place/Transition` nets).

2. Reflective PNs

The *reflective Petri net* approach we have developed [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 [12] 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 (colored) *marking* of the reflective framework, and is automatically updated every time the base level Petri net enters a new state. The reification is used by the meta-program (in the specific by the evolutionary strategies) 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. The occurrence of such one event is modeled by putting a token in a reflective framework's place.

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 to be emulated, 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. We adopted a syntax inspired by Hoare's CSP [10], enriched with a few specific constructs and notations for easy manipulation of nets. A strategy specified

in this way can be automatically translated into the 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. We realistically assume that several strategies are possible at a given instant: the adopted policy is to non-deterministically select one among the ones suitable to be executed. A priority level can be also assigned to alternative strategies, to reduce non determinism.

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 to be put into action;

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

The whole reflective architecture is characterized by a fixed part (the high-level PN representing the reflective framework), and by a part varying from time to time (the base-level PN and the high-level PN representing the evolutionary strategy). The fixed part is used to put evolution into practice for any kind of system, independently of its structure and behavior. It is responsible for the reflective behavior of the architecture, and hides 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 will be updated only when necessary. So the base-level PN model is not polluted by details related to evolution.

3. Transferring a Workflow Instance to a New Template

A workflow management system supporting dynamic workflow change can either directly modify the affected instance, or restart it based on the new workflow template while minimizing repetitive execution of affected nodes. The first method is instance based while the second is template based (schema evolution).

An interesting solution to facilitate efficient dynamic workflow change is proposed in [15]. The approach addresses template-based dynamic workflow changes, according to the consolidated industrial practice that each workflow instance is initiated from its template. The method is implemented in SmarTeam, a leading PDM system with built-in workflow capabilities. Workflows are formally specified in [15] by Directed Network Graphs (DNG), that may be easily translated into PNs.

The idea is to identify all nodes in the new workflow instance that satisfy the following conditions i) they are unchanged, ii) they have been finished in the old workflow instance, and iii) they need not be executed again in the new workflow instance, i.e., they can be bypassed. We hereafter assume that nodes (transitions in PN parlance) subject to change preserve name.

Two nodes are identical before and after change if they represent identical tasks and preserve connections (incoming and outgoing). 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 whose associated task was finished in the old instance can be bypassed when the instance is transferred to a new template, the following additional condition is needed: all nodes from which there is a path (i.e, that are causally connected) to the node itself, can be bypassed. Obviously the start node can be always bypassed. The approach relies upon a simple, clever algorithm for recognizing which nodes can be bypassed: starting from the start node, it only tests the nodes close to a node that can be bypassed. By exploiting the topology of the workflow instance under change, unnecessary tests on non-bypassable nodes are avoided.

The proposed approach has been implemented in Dassault SmarTeam PDM system[1]. SmarTeam has an individual workflow component, which includes a workflow manager, a flowchart designer, and a message management tool. There is no built-in mechanism to support dynamic workflow change. A set of API enables detaching and attaching operations between processes and workflow templates. As earlier indicated, a process has to be re-executed entirely if its workflow template is changed. To realize dynamic workflow change, the proposed approach is implemented as a C++ program, running at the SmarTeam server side. The program executes the following steps:

❶ obtain a flow process instance;

❷ obtain the old and new workflow templates of the flow process;

[1] `http://www.3ds.com/products-solutions/plm-solutions/enovia-smarteam/overview/`.

❸ attach the new workflow template to the flow process;

❹ identify and mark the nodes that can be bypassed in the new workflow instance; and

❺ initiate the new workflow instance without re-executing the marked nodes.

What appears unspecified (or at least, underspecified) in [15] is how to safely operate steps ❹ and ❺: our impression is that some heuristics are implemented, rather than a well defined methodology. No check is done about soundness of changes carried out on workflow instances according to the new template. The latter is assumed to be sound.

4. Reflective PN Alternative to [15]'s Approach

We propose an alternative, based on reflective PNs [5], to the Qiu-Wong [15] approach that allows a sounder formalization of evolutionary steps ❹ and ❺, and permits some kind of on-the-fly verification of workflow changes during evolutionary strategy execution. In case of a negative check, changes are not reflected down to the base-level.

The aim is to show that reflective Petri nets can adequately model adaptable dynamic workflows. We consider the same case presented in [15] (Figure 1), with a small variation. A company has several regional branches. To enhance operation consistence, the company headquarter (HQ) standardizes its business processes in all branches. A process is developed to handle customer problems. The process is defined as a workflow template in SmarTeam. When the staff in a branch encounters a problem, a workflow instance will be initiated from the template and executed until its 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 that are 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 in the HQ database. The HQ manages all instances related to the problem handling process. In response to a business need, the HQ may decide to change the problem handling process in all branches and to transfer all of old workflow instances to the new template, depicted in Figure 1(b). The new template differs from the original one basically in two points:

— "reporting" and "problem solving" are completely separated tasks; and

— it is assumed that "on site realization" can fail, in that case the "problem solving" sequence restarts.

The evolutionary schema. When using reflective PN, the evolutionary schema must be completely reviewed, while preserving motivation and goals of [15] . The new workflow template is not passed as input to the staff of the company branches, rather it results from applying an evolutionary strategy to a workflow instance belonging to the current template. The base-level PN specifying the current workflow instance is hereafter assumed to be *a workflow net* [18], a subclass of 1-bounded `Place/Transition` Petri nets.

It should be fairly evident how simple the design of the base workflow template is. No details related to the net dynamic evolution are hard-wired in the base-level net. Evolution is delegated to the meta-program, translated to a Colored Petri Net, that acts on the reification of base-level workflow. The meta-program is activated when an evolutionary signal is received from HQ, or some anomaly is revealed by introspection (e.g., a deadlock). Introspection is used especially to discriminate whether the evolutionary commands can be safely applied to the current workflow instance, or they must be delayed until its completion (this arbitrary choice is adopted for simplicity).

Figure 1 depicts the following situation: a workflow instance running according to 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") are finished. 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 modified instance, running according to the new workflow template, is illustrated in figure 1(b)).

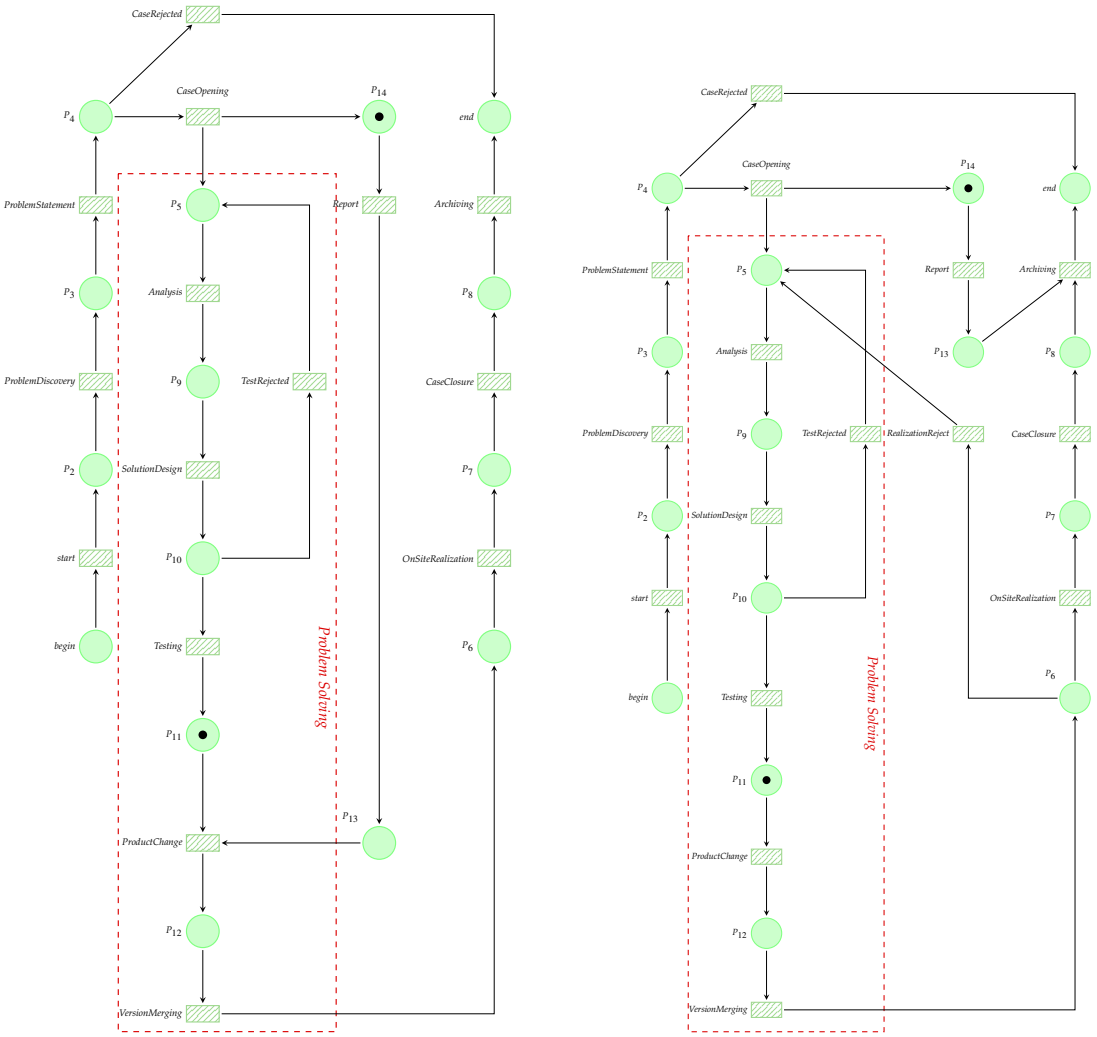
From the above description, one might think that this approach is instance-based rather than template-based. In truth it covers both: if the same evolutionary commands are broadcast to all workflow instances (that change accordingly) we fall in the second category.

The evolutionary strategy, whose formalization is given next, relies upon the definition of *connection preserving* node, a little more flexible than the *unchanged* node notion used in [15]. In some sense it is inspired by van der Aalst's general concept that dynamic workflow change must preserve the inheritance relationship between old and new workflow templates [19].

Let t and t' denote a transition (task) before and after change. Let OLD_NODE be the set of nodes of the old workflow. As usual ${}^\bullet n, n^\bullet$ denote the pre/post sets of a `P/T` node, respectively.

Definition 1 *t is connection preserving if*

$$\forall p \in {}^\bullet t \; \exists p' \in {}^\bullet t', \; {}^\bullet p = {}^\bullet p' \cap OLD\_NODE \wedge$$



(a) The Petri net modeling the base workflow template

(b) A possible workflow evolution

Figure 1. A basic template and its possible evolution.

$$\forall p \in t^\bullet \, \exists p' \in t'^\bullet, \, p^\bullet = p'^\bullet \cap OLD\_NODE \wedge$$
$$\forall p' \in {}^\bullet t' \, \exists p \in {}^\bullet t, \, {}^\bullet p = {}^\bullet p' \cap OLD\_NODE \wedge$$
$$\forall p' \in t'^\bullet \, \exists p \in t^\bullet, \, p^\bullet = p'^\bullet \cap OLD\_NODE$$

Definition 1 is illustrated in figure 2 where the black box denotes a new node. Change (b) "preserves" the old input connections of node t (output connections are unchanged): i.e., whenever the occurrence of t is made possible by the occurrence of some old tasks preceding it, that should happen also in the new situation. That is no more true in case (c): the occurrence of the new node represents now a necessary precondition for any occurrence of t.

Below is the algorithm identifying the non-bypassable nodes:

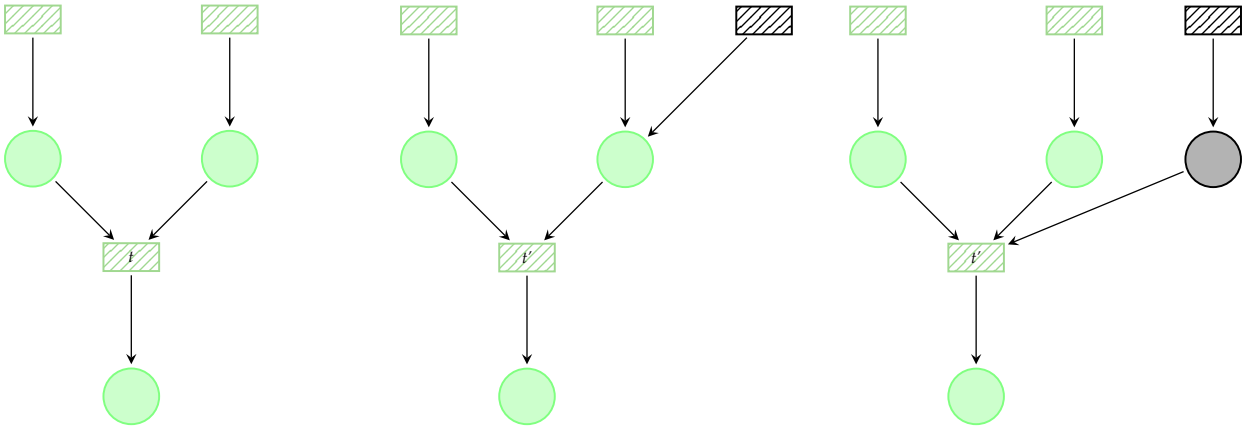
— old transitions whose connections from/to any existing nodes have changed cannot be bypassed;

— let $\langle t, p \rangle$ be an arc such that both p and t are newly added nodes: then $p^\bullet \cap OLD\_NODE$ cannot be bypassed;

— let $\langle p, t \rangle$ be an arc such that both p and t are newly added nodes: then ${}^\bullet p \cap OLD\_NODE$ cannot be bypassed;

— if n cannot be bypassed then all old nodes that n is causally connected to in the new workflow cannot be bypassed.

The meta-program corresponds to the CSP-like code in listing 1, language built-ins and routine calls are in bold.



(a) initial situation (b) change connection preserving (c) change not preserving connections

Figure 2. Definition 1 illustrated.

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 gets activated. Instead if there is an incoming evolutionary message from HQ, the evolutionary strategy starts running. Just after the evolutionary signal, HQ communicates the workflow nodes/connections to be removed and/or added. For sake of simplicity we assume that change can only involve workflow topology. The proposed schema might be adopted as a template for a class of arbitrarily complex evolutionary patterns.

After operating the evolutionary commands on the current workflow reification, the set of non-bypassable nodes is computed on the newly changed reification. Following, the strategy checks through 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 currently pending. In either case a restart procedure takes the workflow reification back to the state before strategy's activation. Otherwise, change is implemented at the base-level through reflection (shift-down). To avoid otherwise possible inconsistencies, the base-level model is "frozen" during strategy's execution. A more sophisticated solution would be that of freezing strategy's influence area on the BL only.

As concerns our reference Petri net class (1-bounded P/T Petri nets), transition t_1 is causally connected to t_2, i.e. its occurrence can favor t_2's occurrence, iff $(t_1{}^\bullet \setminus {}^\bullet t_1) \cap {}^\bullet t_2 \neq \emptyset$. Relation's transitive closure is considered. 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 that initializes the set of non-bypassable nodes (**notBypass**), according to definition 1.

Let us explain how the strategy works considering again Figure 1. Assume that, upon receiving change 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 turn out to be: `Report`, `Archiving`, `ProductChange` (definition 1) and `OnSiteRealization`, `CaseClosure` (causal connection). In the new workflow instance tasks `Report` and `ProductChange` are pending, i.e., they are enabled in the current marking $M : \{p_{11}, p_{14}\}$ of the net in figure 1(b). All old finished tasks that are causally connected to one of them can be bypassed, so the new workflow has not to be restarted, thus saving a lot of work already done.

Assume as a counter example that the only pending task is `OnSiteRealization` (current state $M' : \{p_6\}$), meaning that, among other, `ProductChange`, `VersionMerging` and `Report` are finished in the old workflow instance: in that case change is delayed to the instance completion.

If the suggested change were carried out without consistency control a deadlock would be eventually entered (marking $\{p_8\}$ in figure 1(b)) by the workflow instance running on the modified template.

```
*[
   VAR p, t, n : NODE;
   VAR NEW_PLACE, NEW_TRAN, OLD_NODE={},
       NOT_BYPASS, DEL_NODE: SET(NODE);
   VAR NEW_ARC, DEL_ARC: SET(ARC);

   HQ ? evolution-msg() →
      HQ ? NEW_PLACE; HQ ? NEW_TRAN; HQ ? NEW_ARC;
      HQ ? DEL_ARC; HQ ? DEL_NODE;
      // old workflow reification nodes
   *(n in NODE) [
        exists n → OLD_NODE = OLD_NODE ∪ n
   ]
      // the change is carried out on workflow reification
   newNode(NEW_PLACE ∪ NEW_TRAN); newArc(NEW_ARC);
   deleteArc(DEL_ARC); delNode(DEL_NODE);
     // nodes initially set non-bypassable
   NOT_BYPASS = notBypass()\{start};
     // causally connected nodes are non-bypassable
   NOT_BYPASS = ccTo(NOT_BYPASS) ∩ OLD_NODE;
     // there might be a deadlock, or an old task might be
     // currently enabled due to a finished non-bypassable one
   not(exists t in Tran, enab(t)) or
      (exists t in Tran ∩ OLD_NODE, enab(t) and
      not(isempty(ccBy(t) ∩ NOT_BYPASS))) →
          restart(); // no change actually performed
   // otherwise, change is reflected down to the base-level
   shiftDown()
   □
     #end=0 and not(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 : NODE; RESULT = {} : SET(NODE);

  *(t in OLD_NODE\DEL_NODE ∩ Tran)
    [
      exists p in Place,
       not(isempty({⟨p, t⟩, ⟨t, p⟩} ∩
             (DEL_ARC ∪ NEW_ARC)))
         → RESULT = RESULT ∪ t
    ];

  *(t in NEW_TRAN, p in NEW_PLACE)
    [
      ⟨t, p⟩ in NEW_ARC →
        RESULT = RESULT ∪ post(p) ∩ OLD_NODE;
      □
      ⟨p, t⟩ in NEW_ARC →
        RESULT = RESULT ∪ pre(p) ∩ OLD_NODE;
    ]
]
```

Listing 2. piece of code recognizing the tasks that cannot be bypassed

The mechanism just described ensures a dependable evolution of workflow instances, while being enough flexible. Our aim, however, is not to propose here a general solution to the problem addressed in [15]. Better, sounder and more effective policies do probably exist. Rather, we aim at showing how the approach merging consolidated reflection concepts to classical PN formalisms can be suitably adopted to cope with critical issues related to dynamic workflow change. Besides deadlock freeness, it would be possible to check on-the-fly other properties typical of workflow Petri nets (the basic one is soundness, that is, liveness of tasks plus proper termination), discarding suggested changes if necessary.

Structural base-level analysis. The base-level PN may be analyzed using different techniques. Structural techniques, in particular, are elegant, sound, very efficient, but in general are highly affected by model complexity. Keeping evolutionary aspects separated from functional aspects encourages the use of such techniques.

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

5. Conclusions and Future Work

Covering dynamic and evolutionary aspects of workflow management systems has been widely recognized as a crucial challenge. Consequently there is a high demand for formalisms/tools supporting dynamic workflow design. PNs are a central formalism for modeling workflows, but traditionally they have a static topology. We have proposed and discussed the adoption of a reflective PN-based approach as possible model for workflows prone to be evolved. The model of the current/base workflow behavior, and the model of its evolution, are kept separated, granting therefore the opportunity of analyzing the model without useless details. As application example, an algorithm is delivered to soundly transfer workflow instances from an old to a new template.

Ongoing research is in two different directions. We are

planning to extend the GreatSPN tool for supporting our approach, both in the design and in the analysis phases. We are consolidating the capability of on-the-fly verification of workflow properties, ensuring dependable workflow evolution, sketched in this work.

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] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.