Chapter 10
Trying Out Reflective Petri Nets on a Dynamic Workflow Case

Lorenzo Capra
Università degli Studi di Milano, Italy

Walter Cazzola
Università degli Studi di Milano, Italy

ABSTRACT

Industrial/business processes are an evident example of discrete-event systems which are subject to evolution during life-cycle. The design and management of dynamic workflows need adequate formal models and support tools to handle in sound way possible changes occurring during workflow operation. The known, well-established workflow's models, among which Petri nets play a central role, are lacking in features for representing evolution. We propose a recent Petri net-based reflective layout, called Reflective Petri nets, as a formal model for dynamic workflows. A localized open problem is considered: 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 a workflow management system. Our approach is formal, may be generalized, and is based on the preservation of classical Petri nets structural properties, which permit an efficient characterization of workflow's soundness.

INTRODUCTION

Business processes are frequently subject to change due to two main reasons (Aalst & Jablonski, 2000): 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.

Workflow management facilitates creating and executing business processes. Most of existing Workflow Management Systems, WMS in the sequel (e.g., IBM Domino, iPlanet, Fujisu iFlow, TeamCenter), are designed to cope with static processes. The commonly adopted policy is that, once process changes occur, new workflow templates are defined and workflow instances are initiated accordingly

DOI: 10.4018/978-1-60566-774-4.ch010

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 management might be brought in as a solution. Formal techniques and analysis tools can support the development of WMS able to handle undesired results introduced by dynamic change. Evolutionary workflow design is a challenge on which lot of research efforts are currently devoted. A good evolution is carried out through the evolution of workflow's design information, and then by propagating these changes to the implementation. This approach should be the most natural and intuitive to use (because it adopts the same mechanisms adopted during the development phase) and it should produce the best results (because each evolutionary step is planned and documented before its application).

At the moment evolution is emulated by directly enriching original design information with properties and characteristics concerning possible evolutions. This approach has two main drawbacks: i) all possible evolutions are not always foreseeable; ii) design information is polluted by details related to the design of the evolved system.

In the research on dynamic workflows, the prevalent opinion is that models should be based on a formal theory and be as simple as possible. In Agostini & De Michelis, 2000 process templates are provided as 'resources for action' rather than strict blueprints of work practices. May be the most famous dynamic workflow formalization, the ADEPTflex system (Reichert & Dadam, 1998), is designed to support dynamic change at runtime, making at our disposal a complete and minimal set of change operations. The correctness properties defined by ADEPTflex are used to determine whether a specific change can be applied to a given workflow instance or not.

Petri nets play a central role in workflow modeling (Salimifard & Wright, 2001), due to their description efficacy, formal essence, and the availability of consolidated analysis techniques. Classical Petri nets (Reisig, 1985) have a fixed topology, so they are well suited to model workflows matching a static paradigm, i.e., processes that are finished or aborted once they are initiated. Conversely, any concerns related to dynamism/evolution must be hard-wired in classical Petri nets and bypassed when not in use. That requires some expertise in Petri nets modeling, and might result in incorrect or partial descriptions of workflow behavior. Even worst, analysis would be polluted by a great deal of details concerning evolution.

Separating evolution from (current) system functionality is worthwhile. This concept has been recently applied to a Petri net-based model (Capra & Cazzola, 2007b), called Reflective Petri nets, using reflection (Maes, 1987) as mechanisms that easily permits separation of concerns. A layout formed by two causally connected levels (base-, and meta-) is used. the base-level (an ordinary Petri net) is unaware of the *meta-level* (a high-level Petri net).

Base-level entities perform computations on the entities of the application domain whereas entities in the meta-level perform computations on the entities residing on the lower level. The computational flow passes from the base-level to the meta-level by intercepting some events and specific computations (*shift-up* action) and backs when the meta-computation has finished (*shift-down* action). Meta-level computations are carried out on a representative of the lower-level, called *reification*, which is kept causally connected to the original level.

With respect to other dynamic Petri net extensions (Cabac, Duvignau, Moldt, & Rölke, 2005; Hoffmann, Ehrig, & Mossakowski, 2005; Badouel & Oliver, 1998; Ellis & Keddara, 2000; Hicheur, Barkaoui, & Boudiaf, 2006), Reflective Petri nets (Capra & Cazzola, 2007b) are not a new Petri net

class, rather they rely upon classical Petri nets. That gives the possibility of using available tools and consolidated analysis techniques.

We propose Reflective Petri nets as formal model supporting the design of sound dynamic workflows. A structural characterization of sound dynamic workflows is adopted, based on Petri net's free-choiceness preservation. The approach is applied 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. The problem is efficiently but rather empirically addressed in Qiu & Wong, 2007, according to a template-based schema relying on the concept of *bypassable* task. Conforming to the same concept we propose an alternative, that allows evolutionary steps to be soundly formalized, and basic workflow properties to be efficiently verified.

As widely agreed (Agostini & De Michelis, 2000), the workflow model is kept as simple as possible. Our approach has some resemblance with Reichert & Dadam, 1998, sharing some completeness/smallness criteria, even if it 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" validation.

The balance of the chapter is as follows: first we give a few basic notions around Petri nets and workflows; then we sketch a template-based dynamic workflow approach (Qiu & Wong, 2007) adopted by an industrial WMS; finally, we present our alternative based on Reflective Petri nets, using the same application case as in Qiu & Wong, 2007; we conclude drawing conclusions and perspectives. We refer to the companion chapter (Capra & Cazzola, 2009) for a complete, up-to-date introduction on Reflective Petri nets.

WORKFLOW PETRI NETS

This section introduces the base-level Petri net subclass used in the sequel, with related notations, and properties. We refer to Reisig, 1985; Aalst, 1996 for more elaborate introductions.

Definition 1 (Petri net). *A Petri net is a triple (P;T;F), in which:*

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

In accordance with the simplicity assumption (Agostini & De Michelis, 2000), we are considering a restriction of base-level Petri nets used in Capra & Cazzola, 2009. In the workflow context, it makes no sense to have weighted arcs, because tokens in places correspond to conditions. Consequently, in a well-defined workflow a marking **m** is a set of places, i.e., $\mathbf{m} \in Set(P)$. In general a marking is a mapping, $\mathbf{m} : P \to \mathbb{N}$. Inhibitor arcs and priorities are unnecessary to model the routing of cases in a workflow Petri net.

${}^{\bullet}x, x^{\bullet}$ denote the pre- and post- sets of $x \in P \cup T$, respectively (the set-extensions ${}^{\bullet}\mathrm{A}, \mathrm{A}^{\bullet}\mathrm{A} \subseteq P \cup T$, will be also used). Transitions change the state of the net according to the following rule:

- t is enabled in **m** if and only if each place $p \in {}^{\bullet}t$ contains at least one token.
  - if *t* is enabled in **m** the it *can* fire, consuming one token from each $p \in {}^{\bullet}t$ and producing one token for each $p \in {}^{\bullet}t$

Let $PN = (P;T;F)$, $t_i \in T$, $n_i \in T \cup P$, $\sigma = t_1, t_2, \ldots t_{k-1}$ (possibly $\sigma = \varepsilon$).

BASIC NOTIONS/NOTATIONS

$\mathbf{m}_1 \xrightarrow{t_1} \mathbf{m}_2$ if and only if t_1 is enabled in $\mathbf{m}_1$ and its firing results in $\mathbf{m}_2$

- $\mathbf{m}_1 \xrightarrow{\sigma} \mathbf{m}_k$ if and only if $\mathbf{m}_1 \xrightarrow{t_1} \mathbf{m}_2 \xrightarrow{t_2} \ldots, \mathbf{m}_{k-1} \xrightarrow{t_{k-1}} \mathbf{m}_k$.
- $\mathbf{m}_k$ is *reachable* from $\mathbf{m}_1$ if and only if $\exists \sigma, \mathbf{m}_1 \xrightarrow{\sigma} \mathbf{m}_k$.
- $(PN; \mathbf{m}_0)$ is a Petri net with an initial state $\mathbf{m}_0$.
- Given $(PN; \mathbf{m}_0)$, $\mathbf{m}'$ is said reachable if and only if it is reachable from $\mathbf{m}_0$.

Behavioral Properties

(Live). $(PN; \mathbf{m}_0)$ is live if and only if, for every reachable state $\mathbf{m}'$ and every transition t there exists $\mathbf{m}''$ reachable from $\mathbf{m}'$ which enables t.

(Bounded, safe). $(PN; \mathbf{m}_0)$ is bounded if and only if for each place p there exists $b \in \mathbb{N}$ such that for every reachable state $\mathbf{m}, \mathbf{m}(p) \leq b$. A bounded net is safe if and only if $b = 1$. A marking of a safe Petri net is denoted by a set of places.

Structural Properties

(Path). A path from n_1 to n_k is a sequence $n_1, n_2, \ldots, n_k$ such that $(n_i, n_{i+1}) \in F$, $\forall i$ $1 \leq i \leq k-1$

(Conflict). t_1 and t_2 are in conflict if and only if ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$.

(Free-choice). PN is free-choice if and only if $\forall\, t_1, t_2\ {}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing \Rightarrow {}^\bullet t_1 = {}^\bullet t_2$.

(Causal connection - CC). t_1 is causally connected to t_2 if and only if $(t_1^\bullet \setminus {}^\bullet t_1) \cap {}^\bullet t_2 \neq \varnothing$.

Sound Workflow-Nets and Free-Choiceness

A Petri net can be used to specify the control flow of a workflow. Tasks are modeled by transitions, places correspond to task's pre/post-conditions. Causal dependencies between tasks are modeled by arcs (and places).

Definition 2 (Workflow-net). *A Petri net PN = (P;T;F)* is a Workflow-net (hereafter WF-net) if and only if:

- There is one source place *i* such that ${}^\bullet i = \varnothing$.
- There is one sink place *o* such that $o^\bullet = \varnothing$.
- Every $x \in P \cup T$ is on a path from *i* to *o*.

A WF-net specifies the life-cycle of a case, so it has exactly one input place (*i*) and one output place (*o*). 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 a Petri net. However, there is another requirement that should be satisfied:

Definition 3 (soundness). *A WF-net PN = (P;T;F)* is sound if and only if:

- for every $\mathbf{m}$ reachable from state $\{i\}$, there exists σ, $\mathbf{m} \xrightarrow{\sigma} \{o\}$
- $\{o\}$ is the only marking reachable from $\{i\}$ with at least one token in place *o*
- there are no dead transitions i)n $(PN; \{i\})$, i.e., $\forall t \in T$ there exists a reachable $\mathbf{m}$, $\mathbf{m} \xrightarrow{t} \mathbf{m}'$

In other words: for any cases, the procedure will terminate eventually[1], when the procedure terminates there is a token in place *o* with all the

other places empty (that is referred to as *proper termination*), moreover, it should be possible to execute any tasks by following the appropriate route through the WF-net.

The soundness property relates to the dynamics of a WF-net, and may be considered as a basic requirement for any process. It is shown in Aalst, 1996 that a WF-net *PN* is sound if and only if $(\overline{PN}; \{i\})$ is *live and bounded*. Despite that helpful characterization, deciding about soundness of arbitrary WF-nets may be intractable: liveness and boundedness are decidable, but also EXPSPACE-hard.

Therefore, structural characterizations of sound WF-nets were investigated (Aalst, 1996). Free-choice Petri nets seem to be a good compromise between expressive power and analysis capability. They are the widest class of Petri nets for which strong theoretical results and efficient analysis techniques do exist (Desel & Esparza, 1995). In particular (Aalst, 1996), soundness of a free-choice WF-net (as well as many other problems) can be decided in *polynomial* time. Moreover, a sound free-choice WF-net (PN; $\{i\}$) is guaranteed to be *safe*, according to the interpretation of places as conditions.

Another good reason to restrict our attention to workflow models specified by free-choice WF-nets is that the routing of a case should be independent of the order in which tasks are executed. If non free-choice Petri nets were admitted, then the solution of conflicts could be influenced by the order in which tasks are executed. In literature the term confusion is often used to refer to a situation where free-choiceness is violated by a badly mixture of parallelism and conflict. Free-choiceness is a desirable property for workflows. If a process can be modeled as free-choice WF-net, one should do so. Most of existing WMS support free-choice processes only. We will admit as base-level Petri nets free-choice WF-nets.

Even though free-choice WF-nets are a satisfactory characterization of well-defined workflow procedures, for which soundness can be efficiently checked, there are WF-nets non free-choice which correspond to sensible processes. S-coverability (Aalst, 1996) is a generalization of free-choiceness: a sound free-choice WF-net is in fact S-coverable. In general, it is impossible to verify soundness of an arbitrary S-coverable WF-net in polynomial time, that problem being PSPACE-complete. In many practical cases, however, this theoretical complexity significantly lowers, so that S-coverability could be considered as an interesting alternative to free-choiceness.

A TEMPLATE-BASED APPROACH TO DYNAMIC WORKFLOWS

An interesting solution to facilitate an efficient management of dynamic workflows is proposed in Qiu & Wong, 2007. WMS supporting dynamic workflow change can either directly modify the affected instance, or restart it on a new workflow template. The first method is instance based while the second is template based. The approach we are considering, in accordance with a consolidated practice, falls in the second category, and is implemented in Dassault Systèmes SmarTeam (ENOVIA, 2007), a PLM (Product Lifecycle Management) system including a WMS module. In Qiu & Wong, 2007 workflows are formally specified by Directed Network Graphs (DNG), which can be easily translated into PN.

The idea consists of identifying all *bypassable* tasks, i.e., all tasks 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 re-executed.

A task (transition, in Petri nets) is said *unchanged*, before and after a transformation of the workflow template, if and only if it represents the same activity (what will be always assumed true), and preserves input/output connections. To determine if a task is *bypassable* when the

instance is transferred to a new template, an additional constraint is needed: all tasks from which there is a path (i.e, are causally connected) to the task itself, must be bypassable in turn. A smart algorithm permits the identification of bypassable tasks: starting from the initial task, which is bypassable by default, only successors of bypassable tasks are considered.

This solution has been implemented in SmarTeam system, that includes a workflow manager and a messaging subsystem, but no built-in mechanisms to face dynamic workflow's change. A set of API enables detaching and attaching operations between processes and workflow templates. A process is redone entirely if its template is changed. Workflow's change is implemented by an application-server, which executes the following steps:

1. Obtain a process instance;
2. Obtain the old and new workflow templates;
3. Attach the new workflow template to the process;
4. Identify and mark the tasks that can be bypassed in the new workflow instance;
5. Initiate the new workflow without redoing the marked tasks.

What appears completely unspecified in Qiu & Wong, 2007 is how to safely operate steps 4 and 5: some heuristics appear to be adopted, rather than a well defined methodology. No formal tests are carried out to verify the soundness of a workflow instance transferred to the modified template.

AN ALTERNATIVE BASED ON REFLECTIVE PETRI NETS

We propose an alternative to Qiu & Wong, 2007, based on Reflective Petri nets, which allows a full formalization of the evolutionary steps, as well as a validation of changes proposed for the workflow template, by means of a simple Petri nets structural analysis. Validation is accomplished "on-the-fly", i.e., by operating on the workflow reification while change is in progress. Changes are not reflected to the base-level in case of a negative check. With respect to a preliminary version (Capra & Cazzola, 2007a), the evolutionary strategy, as concerns in particular the validation part, is redesigned and some bugs are fixed.

We consider the same application case presented in Qiu & Wong, 2007. A company has several regional branches. To enhance operation consistence, the company headquarter (HQ) standardizes 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.

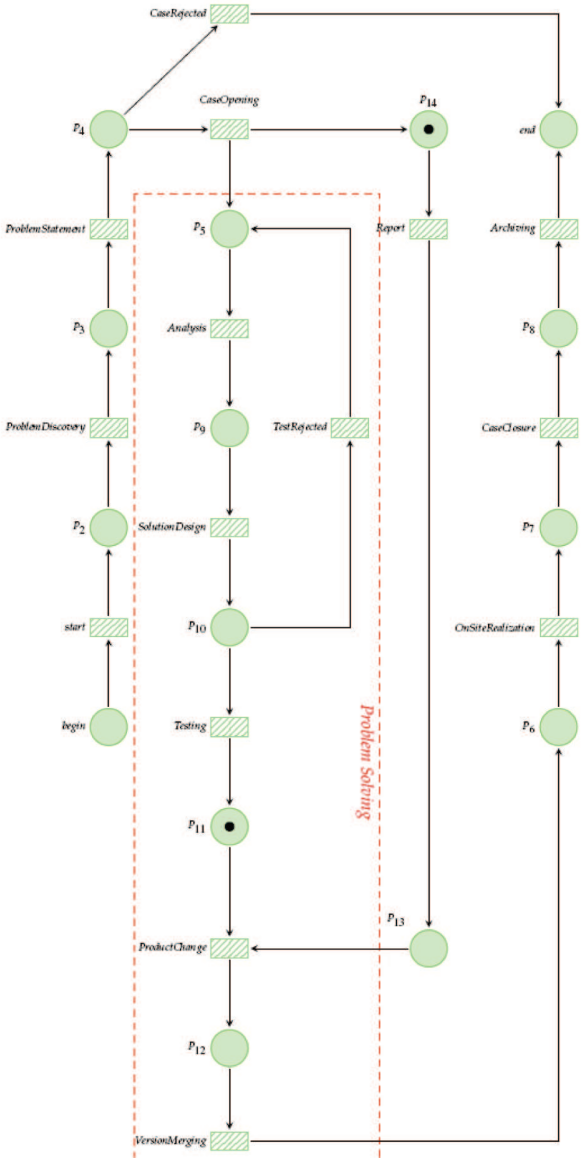
The Petri net specification of the initial template is given in Figure 1. A problem goes through two stages: problem solving and on-site realization. Problem solving involves several tasks, included in a dashed box. When opening a case, the staff reports the case to HQ. When closing the case, it archives the related documents. The HQ manages all instances related to the problem handling process.

In response to business needs, HQ may decide to change the problem handling template. The new template (Figure 2) differs from the original one in two points: a) "reporting" and "problem solving" become independent activities; b) "on site realization" can fail, in that case procedure "problem solving" restarts.

At Petri net level, we can observe that transition Report is causally-connected to ProductChange in Figure 1, while it is not in Figure 2, and that a new transition has been added in Figure 2 (RealizationRejected) which is in free-choice conflict with OnSiteRealization.

When using Reflective Petri nets, the evolutionary schema has to be redesigned. The new workflow template is not passed as input to the staff of the company branches, but it results from

Figure 1. An instance of a workflow template (begin, end are used instead of i and o)

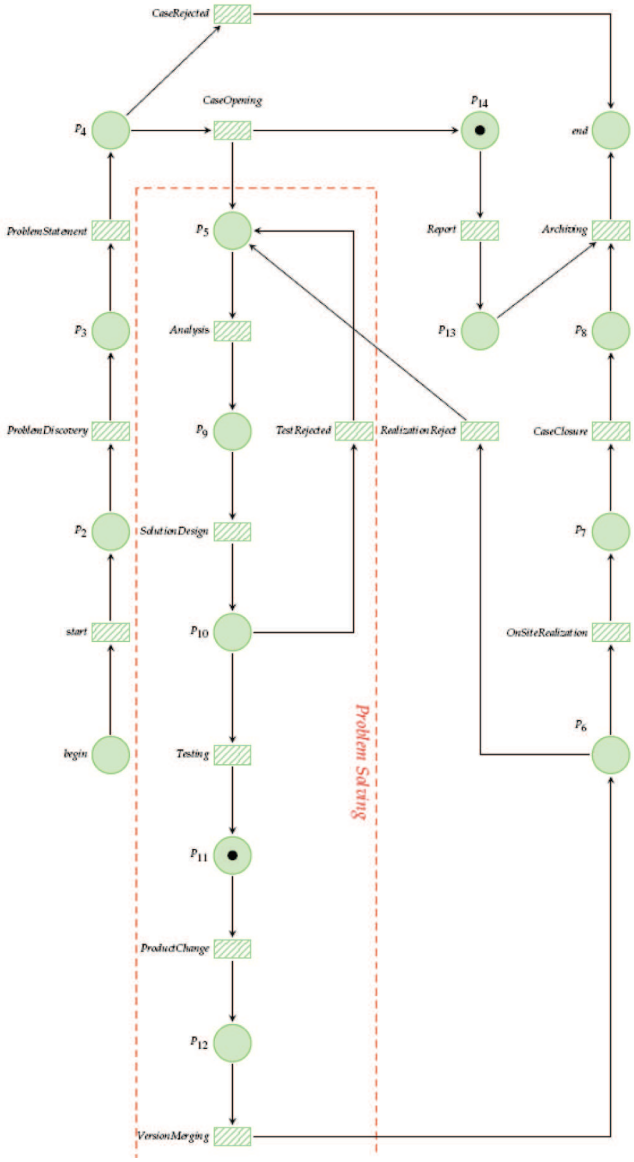


applying an evolutionary strategy to a workflow instance belonging to the current template. The initial base-level Petri nets is assumed a *free-choice WF-net*. No details about the workflow dynamics are hard-wired in the base-level net. Evolution is delegated to the meta-program, that acts on the WF-net reification.

The meta-program is activated when an evolutionary signal is sent in by HQ, or some anomaly (e.g., a deadlock) is revealed by introspection. (Late) introspection is also used to discriminate whether evolutionary commands have been safely applied to the current workflow instance, or they have to be discarded.

Figure 1 depicts the following situation: a workflow instance running on the initial template has received a message from HQ. At the current state (marking) SolutionDesign, a sub-task of

Figure 2. Workflow's evolution



ProblemSolving, and Report are pending tasks, whereas a number of tasks (e.g., Analysis and CaseOpening) have been completed. The meta-program in that case successfully operates a change on the old template's instance, once verified that all paths to any pending tasks are only composed of bypassable tasks.

The workflow instance transferred to the new template is illustrated in Figure 2.

One might think of this approach as instance-based, rather than template-based. In truth it covers both: if the evolutionary commands are in fact broadcasted to workflow's instances we fall in the latter scheme.

The evolutionary strategy relies upon the notion of *adjacency preserving* task, which is more general than the *unchanged* task used in Qiu & Wong, 2007. It is inspired by van der Aalst's

concept that any workflow change must preserve the inheritance relationship between old and new templates (Aalst & Basten, 2002). Let us introduce some notations used in the sequel.

Let $PN = (Old\_P; Old\_T; Old\_A)$, be a base-level WF-net (better, its reification at the meta-level), $PN' = (P'; T'; A')$ be the resulting Petri net after some modifications[2], $Old\_N = Old\_P \cup Old\_T$, $N' = P' \cup T'$.

Symbols x and $\overline{x}$ refer to a node "preserved" by change, considered in the context of PN and PN', respectively.

Sets $Del\_N = Del\_P \cup Del\_T$, $New\_N = New\_P \cup New\_T$, and New_A, Del_A, denote the base level nodes/arcs added to and removed from PN, respectively.

We assume that $New\_A \cap Del\_A = \varnothing$. No other assumptions are made: for example "moving" a given node across the base-level Petri net might be simulated by first deleting the node, then putting it again setting new connections.

As explained in (Capra & Cazzola, 2009), the evolutionary framework (a transparent meta-level's component) being in charge of carrying out evolution rejects a proposed change if not consistent with respect to the current base-level's reification.

Finally, NO_ADJ, NO_BYPS denote the tasks not preserving adjacency and the non-bypassable tasks, respectively (of course, $NO\_ADJ \subseteq NO\_BYPS$). Some of the symbols just introduced will be used as names for the evolutionary strategy parameters.

Definition 4 (adjacent set). *Let t be a transition. The set of adjacent transitions A_t is:*

$$^\bullet(^\bullet t \cup t^\bullet) \cup (^\bullet t \cup t^\bullet)^\bullet \setminus \{t\}.$$

Definition 5 (adjacency preserving task). *Let* $t \in Old\_T$, $\overline{t} \in T'$. Task t is adjacency preserving if and only if $\forall x \in Old\_T, \overline{x} \in A_t \Leftrightarrow x \in A_t$ and there exist a bijection $\phi : {}^\bullet t \cup t^\bullet \to {}^\bullet\overline{t} \cup \overline{t}^\bullet$ such that

$\forall x \in A_t \forall y \in {}^\bullet t \cup t^\bullet$, $y \in {}^\bullet x \Leftrightarrow \phi(y) \in {}^\bullet\overline{x}$ and $y \in x^\bullet \Leftrightarrow \phi(y) \in \overline{x}^\bullet$

If t is adjacency preserving then all its causality/conflict relationships to adjacent tasks are maintained. A case where Definition 5 holds, and another one where it does not, are illustrated in Figure 3 (the black bar denotes a new task, *t'* is used instead of $\overline{t}$). In case (b) the original input connections of t are maintained (output connections are unchanged): if the occurrence of t is made possible by the occurrence of some preceding tasks it, the same may happen in the new situation. That is not true in case (c): the occurrence of the new task represents in fact an additional precondition for any subsequent occurrence of t.

Checking definition 5 is computationally very expensive. However, if useless changes are forbidden, e.g., "deleting a given place p, then adding p' inheriting from p all connections", or "adding an arc $\langle p, t \rangle$, then deleting p or t", check's complexity can be greatly reduced.

Lemma 1 states some rules for identifying a *superset* of tasks N_a not preserving adjacency. It can be easily translated to an efficient meta-program's routine. Almost always it comes to be $N_a \equiv NO\_ADJ$.

Lemma 1. *Consider set* N_a, built as follows

$$p \in Del\_P \Rightarrow {}^\bullet p \cup p^\bullet \subseteq N_a$$

$$t \in Del\_T \Rightarrow {}^\bullet({}^\bullet t) \cup (t^\bullet)^\bullet \subseteq N_a$$

$$\langle p,t \rangle \in Del\_A \vee \langle t,p \rangle \in Del\_A \Rightarrow {}^\bullet p \cup p^\bullet \subseteq N_a$$

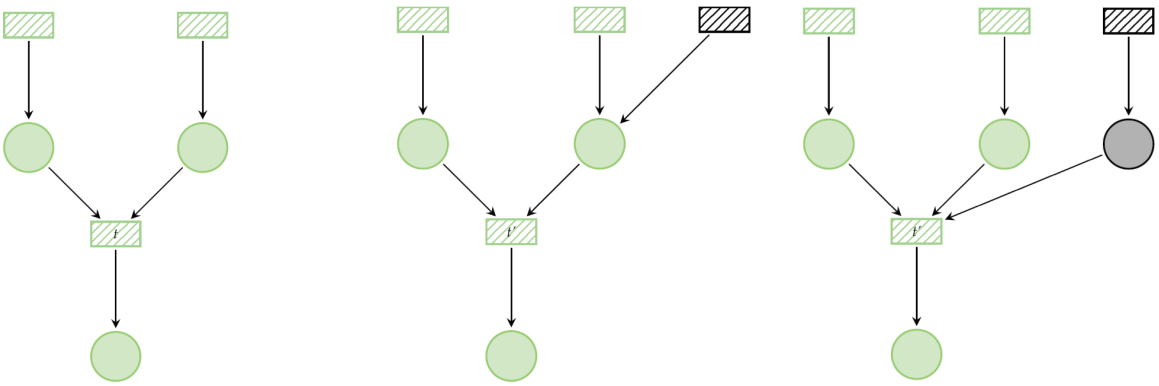
$$\langle p,t \rangle \in New\_A \wedge t \in Old\_N \Rightarrow \{t\} \cup D \subseteq N_a$$

where $D = {}^\bullet p \cup p^\bullet$ if $p \in Old\_N$, else $D = \varnothing$

Then $NO\_ADJ \subseteq N_a$.

The evolutionary meta-program if formalized in Figure 4. The use of a CSP-like syntax (Hoare, 1985; Capra & Cazzola, 2009) makes it possible its automatic translation to a high-level Petri net

Figure 3. Definition 5 Illustrated



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

(the logical meta-level of the Reflective Petri nets layout). 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.

```
*[
  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);
  //receiving an evolutionary signal
  HQ ? change-msg() -> [
  //receiving the evolutionary commands
  HQ ? New_P; HQ ? New_T; HQ ? New_A; HQ
? Del_A; HQ ? Del_N;
  //getting the WF-net reification
  Old_N = ReifiedNodes();
  //computing the non-bypassable tasks
  NO_BYPS = ccTo(notAdjPres());
//changing the current reification
  newNode(New_P ∪ New_T);newArc(New_A);
deleteArc(Del_A); delNode(Del_N);
  //checking the (new) WF-net's well-def-
initeness
  checkWfNet(); checkFc();
  /*there might be a deadlock, or a non-
bypassable task is causally
  Connected  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)) -> [re-
start()] //rejecting changeshiftDown() //
reflecting change
  ]
[]
  #end=0 and !exists t in Tran, enab(t)
-> [HQ ! notify-deadlock()]
[]
  #end=1; HQ ? newInstance-msg() ->
[flush(end); incMark(begin)]
]
```

Just after an evolutionary signal, HQ communicates the workflow nodes/connections to

Figure 4. Workflow's evolutionary strategy

```
*[
  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);
  //receiving an evolutionary signal
  HQ ? change-msg() -> [
  //receiving the evolutionary commands
  HQ ? New_P; HQ ? New_T; HQ ? New_A; HQ ? Del_A; HQ ? Del_N;
  //getting the WF-net reification
  Old_N = ReifiedNodes();
  //computing the non-bypassable tasks
  NO_BYPS = ccTo(notAdjPres());
  //changing the current reification
  newNode(New_P∪New_T);newArc(New_A);                deleteArc(Del_A);
delNode(Del_N);
  //checking the (new) WF-net's well-definiteness
  checkWfNet(); checkFc();
  /*there might be a deadlock, or a non-bypassable task is causally
  Connected  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()] //rejecting change
  shiftDown() //reflecting change
  ]
[]
  #end=0 and !exists t in Tran, enab(t) -> [HQ ! notify-deadlock()]
[]
  #end=1; HQ ? newInstance-msg() -> [flush(end); incMark(begin)]
]
```

be removed/added. For the sake of simplicity we assume that change can only involve workflow's topology. The (super)set of non-bypassable tasks is then computed.

After operating the evolutionary commands on the current workflow reification, definition 2 and free-choiceness are tested 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). The scheme just described might be adopted for a wide class of evolutionary patterns.

Language's keywords and routine calls are in bold. We recall (Capra & Cazzola, 2009) that type **NODE** represents a (logically unbounded) recipient of base-level nodes, and is partitioned into **Place** and **Tran** subtypes . 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 tasks to N_a according to lemma 1. The routines **ccTo** and **ccBy** compute the set of nodes the argument is causally connected to, and that are causally connected to routine's argument, respectively.

On-the-Fly Structural Check

The structural changes proposed from time to time to a dynamic workflow can be validated by means of classical Petri nets analysis techniques. Validation is accomplished on the workflow reification "on-the-fly", i.e., while the evolutionary strategy is in progress. Thanks to a restart mechanism, potentially dangerous changes are discarded before they are reflected to the base-level, at the end of a meta-computation.

Routines **checkWfNet**, **checkFc** test the preservation of base-level Petri nets well-definiteness (definition 2) and free-choiceness, respectively. Their calls are located in the meta-program just after the evolutionary commands, which operate on the base-level workflow reification.

```
[
VAR t,tx: Tran;
VAR p: Place;
*(<p,t> in New_A ∪ Del_A)
  [
   exists(p) and exists(t) ->
     [
      *(tx in post(pre(t))/{t})
       [pre(t) <> pre(tx) -> restart();]
     ]
  ]
]
```

Free-choiceness preservation, in particular, may be checked in a simple, efficient way. Figure 5 expands the corresponding routine. It works under the following assumptions and principles:

- the initial base-level Petri net is a free-choice WF-net (conservative hypothesis)
- variables New_A, Del_A record *all* the arcs which are added/deleted to/from the base-level reification during the evolutionary strategy's execution; they are cleared at any meta-program activation;
- the only operations affecting free-choiceness, under a conservative hypothesis, are the addition/removal of an input arc $\langle p,t \rangle$ (the removal of a node produces as a side-effect the withdrawal of all adjacent arcs, so it is fair, with respect to free-choiceness)
- for each arc $\langle p,t \rangle$ which has been added/removed we only have to check the free-choiceness between t and every transition sharing with t some input places.

Putting the Strategy to the Test

Let us explain how the strategy works, considering again Figures 1-2, upon receiving the evolutionary commands:

New_P={}
New_T={ RealizationRejected}
Del_A={ $\langle p13$, ProductChange$\rangle$ }
Del_N={}
New_A={$\langle p6$, Realization Rejected$\rangle$, $\langle p13$, Archiving$\rangle$, $\langle$Realization Rejected, $p5\rangle$}.

The non-bypassable tasks come to be: {Report, Archiving, ProductChange, OnSiteRealization, CaseClosure}. In the workflow instance running on the modified template (Figure 2) tasks (transitions) Report and ProductChange are pending (enabled) in the current state (marking) **m**: $\{p_{11}, p_{14}\}$. All preexisting 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.

The approach just described ensures a dependable evolution of workflows, while being enough flexible. We have not intended to propose a general solution to the particular problem addressed in Qiu & Wong, 2007. Better policies do probably exist. Rather, we have tried to show that an approach merging consolidated reflection concepts to classical Petri nets techniques can suitably address the criticisms of dynamic workflow change.

Figure 5. Meta-program's routine checking free-choiceness

```
[
 VAR t,tx: Tran;
 VAR p: Place;
 *(<p,t> in New_A∪Del_A)
   [
    exists(p) and exists(t) ->
      [
       *(tx in post(pre(t))/{t})
        [pre(t) <> pre(tx) -> restart();]
      ]
   ]
]
```

Structural Base-Level Analysis. The base-level is guaranteed to be a free-choice WF-net all over its evolution: that makes is possible to use polynomial algorithms to check workflow's soundness. In particular techniques based on the calculus of flows (invariants) are elegant and very efficient. In general they are highly affected by the base-level Petri net complexity. The separation between evolutionary and functional aspects encourages their usage.

For instance, by operating the structural algorithms of GreatSPN tool (Chiola, Franceschinis, Gaeta, & Ribaudo, 1995), it is possible to discover that both nets depicted in Figures 1-2 are covered by *place-invariants*. A lot of interesting properties thereby descend: in particular boundedness and liveness, i.e., workflow soundness.

Counter Example

Assume that evolution occurs when the only pending task is OnSiteRealization, i.e., consider as current marking of the net in Figure 1 $\mathbf{m'}: \{p_6\}$. That means, among the other things, tasks ProductChange, VersionMerging and Report have been completed: change in that case is discarded after having verified that there are some non-bypassable tasks which are causally connected to the pending one. If the suggested change were really carried out (reflected) on the base-level, without doing any consistency control, a deadlock would be eventually entered (state $\{p_8\}$) after the workflow instance continues its run on the modified template. The problem is that $\mathbf{m'}$ is not a reachable state of $(PN'; \{begin\})$, but reachability is NP-complete also in live and safe free-choice Petri nets, so it would make no sense checking reachability directly at meta-program level.

Current Limitations

The proposed reflective model for dynamic workflows suffers from a major conceptual limitation: only the control-flow perspective is considered. Let us shortly discuss this choice. We abstract from the resource perspective because in most workflow management systems it is not possible to specify that several (human) resources are collaborating in executing a task. Even if multiple persons are executing a task, only one is allocated to it from the WMS perspective: who selected the work item from the in-basket. In contrast to other application domains such as flexible manufacturing systems, anomalies resulting from locking problems are not possible (it is reasonable to assume that each task will be eventually executed by the person having in charge it). Therefore, from the viewpoint of workflow verification, we can abstract

from resources. However, if collaborative features will be explicitly supported by WMS (through a tight integration of groupware and workflow technology), then the resource perspective should be taken into account. We partly abstract from the data perspective. Production data can be changed at any time without notifying the WMS. Their existence does not even depend upon the workflow application and they may be shared among different workflows. The control data used to route cases are managed by the WMS. Some of these data are set or updated by humans or applications. Clearly, some abstraction is needed to incorporate the data perspective when verifying a given workflow. The abstraction currently used is the following. Since control data (workflow attributes such as the customer id, the registration date, etc.) are only used for the routing of a case, we model each decision as a non-deterministic choice. If we are able to prove soundness for the situation without workflow attributes, it will also hold for the situation with attributes. Abstracting from triggers and workflow attributes fits in the usage of ordinary Petri nets for the base-level of the reflective model: this is preferable because of the availability of efficient and powerful analysis tools.

CONCLUSION

Industrial/business processes are an example of discrete-event systems which are increasingly subject to evolution during life-cycle. Covering the intrinsic dynamism of modern processes has been widely recognized as a challenge by designers of workflow management systems. Petri nets are a central model of workflows, but traditionally they have a fixed structure. We have proposed and discussed the adoption of Reflective Petri nets as a formal model for sound dynamic workflows. The clean separation between (current) workflow template and evolutionary strategy on one side, and the use of classical Petri nets notions (free-choiceness) on the other side, make it possible to efficiently check preservation of workflow's structural properties, which in turn permit soundness -a major behavioral property- to be checked in polynomial time. All is done while evolution is in progress. An algorithm is delivered to soundly transferring workflow instances from an old to a new template, redoing already completed workflow's task only when strictly necessary. The approach formalizes and improves a procedure currently implemented in an industrial workflow management system. We are studying the possibility of using even more general structural notions than free-choiceness, in particular S-coverability (Aalst, 1996), that provides in most practical cases a structural characterization of soundness.

REFERENCES

Agostini, A., & De Michelis, G. (2000, August). A Light Workflow Management System Using Simple Process Models. *Computer Supported Cooperative Work*, *9*(3-4), 335–363. doi:10.1023/A:1008703327801

Badouel, E., & Oliver, J. (1998, January). *Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems* (IRISA Research Report No. PI-1163). IRISA.

Cabac, L., Duvignau, M., Moldt, D., & Rölke, H. (2005, June). Modeling Dynamic Architectures Using Nets-Within-Nets. In G. Ciardo & P. Darondeau (Eds.), *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)* (p. 148-167). Miami, FL: Springer.

Capra, L., & Cazzola, W. (2007a, on 26th-29th of September). 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)* (p. 533-540). Timisoara, Romania: IEEE CS.

Capra, L., & Cazzola, W. (2007b, December). Self-Evolving Petri Nets. *Journal of Universal Computer Science*, *13*(13), 2002–2034.

Capra, L., & Cazzola, W. (2009). An Introduction to Reflective Petri Nets. In E. M. O. Abu-Atieh (Ed.), *Handbook of Research on Discrete Event Simulation Environments Technologies and Applications*. Hershey, PA: IGI Global.

Chiola, G., Franceschinis, G., Gaeta, R., & Ribaudo, M. (1995, November). GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, *24*(1-2), 47–68. doi:10.1016/0166-5316(95)00008-L

Desel, J., & Esparza, J. (1995). *Free Choice Petri Nets* (Cambridge Tracts in Theoretical Computer Science Vol. 40). New York: Cambridge University Press.

Ellis, C., & Keddara, K. (2000, August). ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems. *Computer Supported Cooperative Work*, *9*(3-4), 293–333. doi:10.1023/A:1008799125984

ENOVIA. (2007, September). *Dassault systèmes plm solutions for the mid-market* [white-paper]. Retrieved from. http:/www.3ds.com/fileadmin/brands/enovia/pdf/whitepapers/CIMdata-DS_PLM_for_the_MidMarket-Program_review-Sep2007.pdf)

Hicheur, A., Barkaoui, K., & Boudiaf, N. (2006, September). Modeling Workflows with Recursive ECATNets. *In Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNACS'06)* (p. 389-398). Timisoara, Romania: IEEE CS.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Upper Saddle River, NJ: Prentice Hall.

Hoffmann, K., Ehrig, H., & Mossakowski, T. (2005, June). High-Level Nets with Nets and Rules as Tokens. In G. Ciardo & P. Darondeau (Eds.), *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)* (pp. 268-288). Miami, FL: Springer.

Maes, P. (1987, October). Concepts and Experiments in Computational Reflection. In *N. K. Meyrowitz (Ed.), Proceedings of the 2nd conference on object-oriented programming systems, languages, and applications (OOPSLA'87)* (Vol. 22, pp.147-156), Orlando, FL.

Qiu, Z.-M., & Wong, Y. S. (2007, June). Dynamic Workflow Change in PDM Systems. *Computers in Industry*, *58*(5), 453–463. doi:10.1016/j.compind.2006.09.014

Reichert, M., & Dadam, P. (1998). ADEPTflex - Supporting Dynamic Changes in Workflow Management Systems without Losing Control. *Journal of Intelligent Information Systems*, *10*(2), 93–129. doi:10.1023/A:1008604709862

Reisig, W. (1985). *Petri nets: An introduction* (EATCS Monographs in Theoretical Computer Science Vol. 4). Berlin: Springer.

Salimifard, K., & Wright, M. B. (2001, November). Petri Net-Based Modeling of Workflow Systems: An Overview. *European Journal of Operational Research*, *134*(3), 664–676. doi:10.1016/S0377-2217(00)00292-7

van der Aalst, W. M. P. (1996). *Structural Characterizations of Sound Workflow Nets* (Computing Science Reports No. 96/23). Eindhoven, the Netherlands: Eindhoven University of Technology.

van der Aalst, W. M. P., & Basten, T. (2002, January). Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, *270*(1-2), 125–203. doi:10.1016/S0304-3975(00)00321-2

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

KEY TERMS AND DEFINITIONS

Evolution: attitude of systems to change layout/functionality.

Dynamic Workflows: models of industrial/business processes subject to evolution.

Petri Nets: graphical formalism for discrete-event systems.

Reflection: activity performed by an agent when doing computations about itself.

Workflow Nets: Petri net-based workflow models.

Soundness: behavioral property of a well-defined workflow net.

Structural Properties: properties derived from the incidence matrix of Petri nets.

Free-Choiceness: typical structural property which can be efficiently tested.

ENDNOTES

1 If we assume, as it is reasonable in the workflow context, a strong notion of fairness: in every infinite firing sequence, each transition fires infinitely often.

2 We recall that in Reflective Petri nets any evolutionary strategy is defined in terms of basic operations on base-level's elements