Fundamentals of Control Flow in Workflows*

B. Kiepuszewski[1], A.H.M. ter Hofstede[1], and W.M.P. van der Aalst[1,2]

[1] *Centre for Information Technology Innovation, Queensland University of Technology*
GPO Box 2434, Brisbane Qld 4001, Australia, e-mail: bkiepuszewski@infovide.pl, a.terhofstede@qut.edu.au;
[2] *Department of Technology Management, Eindhoven University of Technology*
GPO Box 513, NL-5600 MB Eindhoven, The Netherlands, e-mail: w.m.p.v.d.aalst@tm.tue.nl.

Abstract

Although workflow management emerged as a research area well over a decade ago, little consensus has been reached as to what should be essential ingredients of a workflow specification language. As a result, the market is flooded with workflow management systems, based on different paradigms and using a large variety of concepts. The goal of this paper is to establish a formal foundation for control-flow aspects of workflow specification languages, that assists in understanding fundamental properties of such languages, in particular their expressive power. Workflow languages can be fully characterized in terms of the evaluation strategy they use, the concepts they support, and the syntactic restrictions they impose. A number of results pertaining to this classification will be proven. This should not only aid those developing workflow specifications in practice, but also those developing new workflow engines.

Keywords: Workflow management, Expressiveness, Petri nets, Equivalence of behaviour, Control flow.

1 Introduction

Workflow technology continues to be subjected to on-going development in its traditional application areas of business process modelling, business process coordination and document and image management, and now in emergent areas such as business-to-business and business-to-consumer interactions. Addressing this broad and rather ambitious reach, a large number of workflow products, mainly workflow management systems (WFMS), are commercially available [AH02, Fis01, JB96, LR99, Law97]. We have evaluated 15 workflow management systems using a comprehensive set of workflow patterns [ABHK00, AHKB00, AHKB02, WPH02]. This evaluation revealed that contemporary products use a variety of workflow languages resulting in different capabilities. This inspired us to look into the fundamental mechanisms for handling control flow in workflow technology.[1]

*This research is supported by an ARC SPIRT grant "Component System Architecture for an Open Distributed Enterprise Management System with Configurable Workflow Support" between QUT and Mincom.

[1] Part of this introduction is taken from [AHKB02].

Workflow specifications can be understood, in a broad sense, from a number of different perspectives (see [JB96]). The *control-flow* (or process) perspective describes activities and their execution ordering through different constructors, which permit flow of execution control, e.g. sequence, choice, parallelism, and synchronization. Activities in elementary form are atomic units of work, and in compound form modularise an execution order of a set of activities. The *data perspective* layers business and processing data on the control flow perspective. Business documents and other objects which flow between activities, and local variables of the workflow, qualify in effect pre- and post-conditions of activity execution. The *resource perspective* provides an organizational structure anchor to the workflow in the form of human and device roles responsible for executing activities. The *operational* perspective describes the elementary actions executed by activities, where the actions map into underlying applications. Typically, (references to) business and workflow data are passed into and out of applications through activity-to-application interfaces, allowing manipulation of the data within applications.

Clearly, the control flow perspective provides an essential insight into a workflow language's effectiveness. The data flow perspective rests on it, while the organizational and operational perspectives are ancillary. Currently, most workflow languages support the basic constructs of sequence, iteration, splits (AND and OR) and joins (AND and OR) - see e.g. [Fis01, Law97]. However, the interpretation of even these basic constructs is not uniform and it is often unclear how more complex requirements could be supported. Indeed, vendors are afforded the opportunity to recommend implementation level "hacks" resulting in coding outside the workflow management system. The result is that neither current capabilities nor an insight into newer requirements is advanced.

Problem

The distinctive features of different workflow languages allude to fundamentally different semantics. Some languages allow multiple instances of the same activity type at the same time in the same workflow context while others do not. Some languages structure loops with one entry point and one exit point, while in others loops are allowed to have arbitrary entry and exit points. Some languages require explicit termination activities for workflows and their compound activities while in others termination is implicit. Such differences point to different insights of *suitability* and different levels of *expressive power*.

The goal of this paper is to build a formal foundation in which control flow aspects in workflow languages can be comprehensively understood, and to use this foundation to prove a number of fundamental results characterizing the expressive power of various of these languages. This should not only help analysts specifying workflows in practice, as they may have to understand fundamental limits of the workflow language they use or have to map their specifications to, but also developers designing new workflow engines, as the results presented may prevent them from imposing restrictive constraints on workflow specifications, or may, in some cases, provide them with certain useful equivalence preserving transformations. With the increasing maturity of workflow technology, workflow language extensions, we feel, should be levered across the board, rather than slip into "yet another technique" proposals.

Approach

As it turns out, workflow languages can, as far as the control flow perspective goes, be fully characterized in terms of the evaluation strategy they use, the concepts they support, and the syntactic restrictions they impose. Based upon the evaluation strategy, a mapping of workflows to Petri nets (see e.g. [Mur89, RR98]) is presented. Petri nets were chosen as they provide a general, well understood and well researched, theory for concurrency.

Petri nets have been proposed for modelling workflow process definitions long before the term "workflow management" was coined and workflow management systems became readily available. Consider for example the work on Information Control Nets, a variant of the classical Petri nets, in the late seventies [Ell79]. Petri nets constitute a good starting point for a solid theoretical foundation of workflow management. Clearly, a Petri net can be used to specify the control-flow, i.e., the routing of cases (workflow instances) [Aal98]. Activities are modelled by transitions and causal dependencies are modelled by places, transitions, and arcs. In fact, a place corresponds to a condition which can be used as pre- and/or post-condition for activities. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs.

In this paper, we do not use high-level Petri nets such as Coloured Petri nets [Jen87] and Predicate/Transition nets [Gen87] or specialized models such as Bipolar Synchronization Schemes [GT84]. Although high-level nets are a good language for specifying workflows, it is difficult to compare different workflow languages once they are mapped onto high-level nets (the control-flow can be mapped onto the network structure or onto data structures) and many questions become undecidable. Therefore, we provide mappings from various workflow models onto low-level Petri nets (i.e., place/transition nets [Mur89, RR98]). Although we abstract from the data perspective and the resource perspective, we acknowledge the need for research activities dealing with (limited) notions of data and work distribution. For such activities, coloured Petri nets seem to be more appropriate.

The mappings presented, assigning a formal semantics to workflow languages, together with the "right" notion of equivalence, then allow an in-depth investigation into expressiveness properties of various classes of workflow languages.

Outline

The organization of this paper is as follows. First the classification of workflow languages based on their evaluation strategy is discussed as well as the associated mappings to Petri nets (Section 2). Then in Section 3 a discussion of the right notion of equivalence in the context of workflows is provided. In Section 4 a number of basic expressiveness results is presented, while in Section 5 focus is on more advanced expressiveness results. Section 6 concludes the paper and provides a summary of the main results. Appendix A contains definitions and notations pertaining to Petri nets as used in this paper.

2 Formal Foundations

Consideration of a large number of commercially available workflow management systems, some research prototypes, and some workflow languages developed in academia, led us to classify workflow languages in terms of three evaluation strategies used (the interested reader is referred to [ABHK00, AHKB00, AHKB02] for a product description and evaluation of 15 workflow management systems; there is also a WWW-page describing the patterns [WPH02]). It is quite conceivable that products exist that escape this classification and undoubtedly such products may emerge in the future, but as will be shown, these three classes capture current "mainstream" thinking about workflow specifications fairly accurately. Emphasis in this section is on the formal foundations of these classes of workflow languages, as they will allow us to prove fundamental expressiveness results in the rest of this paper.

Since 1993, the Workflow Management Coalition[2] (WfMC) has focused on furthering the field of workflow management by providing standards, common terminology, and interfaces. In this paper, their terminology will be adopted as much as possible. First, the main definitions with respect to basic control flow constructs will be recalled from [Wor99] in order to establish a common understanding important for the discussion on formal foundations following later in this section.

An *AND-split* is "a point within the workflow where a single thread of control splits into two or more threads which are executed in parallel within the workflow, allowing multiple activities to be executed simultaneously". The WfMC additionally observes that *in certain workflow systems all the threads created at an AND-split must converge at a common AND-join point (Block Structure); in other systems convergence of a subset of the threads can occur at different AND-join points, potentially including other incoming threads created from other AND-split points (Free Graph Structure).*

An *AND-join* is "a point in the workflow where two or more parallel executing activities converge into a single common thread of control". In the definition there seems to be an implicit assumption that both parallel threads eventually will "reach" that common point as it is not stated what should happen if this is not the case.

An *OR-split* is "a point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches". Note that this definition implies that only one branch can be taken from all the alternatives (for this reason, to avoid ambiguity, when we mean that only one of the alternative branches can be chosen, we will use the term *XOR-split* in this paper).

Finally, an *OR-join* is "a point within the workflow where two or more alternative activity(s) workflow branches re-converge to a single common activity as the next step within the workflow". In addition it is noted that "as no parallel activity execution has occurred at the join point, no synchronization is required". Again, it is unclear how the OR-join should behave if parallel execution actually *does* occur before the join point.

As will be shown in the following subsections, the lack of formal semantics associated with these constructs has resulted in different interpretations by different vendors thus significantly reducing potential workflow interoperability. We have identified three evaluation strategies,

[2] http://www.wfmc.org

which all have fundamentally different interpretations of the aforementioned basic control constructs:

- Standard Workflow Models (Section 2.1);
- Safe Workflow Models (Section 2.2);
- Synchronizing Workflow Models (Section 2.3).

2.1 Standard Workflow Models

Standard Workflow Models represent what would appear to be the most "natural" interpretation of the WfMC definitions. In this section a mapping of the WfMC basic control flow constructs to Petri nets is provided, which captures this interpretation formally. In addition to the mapping a justification is supplied as to why we think this mapping represents the "intent" of the broader workflow community and a discussion of how it compares to some other mappings which have been proposed.

In the vast majority of workflow management systems when an activity instance is finished, the next activity instance to be executed is selected and its state is changed to `READY` (this typically corresponds to placing it on a designated worklist). After this, the activity instance can go through a number of internal states. Finally, if all the associated processing has been performed successfully, its state is changed to `COMPLETE`. As will be seen, these two states are crucial to control flow considerations and any formal semantics of control flow constructs has to take at least these two states into account explicitly.

When using Petri nets for capturing formal semantics of workflows, there is a choice of labelling places or transitions, where the labels represent activities that are to be performed. We have chosen to label transitions as this appears to be more common. In this approach, a labelled transition being enabled indicates that the corresponding activity is in the `READY` state. Firing the transition then corresponds to executing the activity and changing its state to `COMPLETE`.

Not all transitions are labelled. Transitions without a label (sometimes the label λ is used, representing an internal or "silent" action; in the remainder of this paper we will refer to such transitions as λ-transitions) represent internal processing performed by the workflow engine which cannot be observed by the external users (though they may also represent execution of the so-called null activity, the activity which does nothing). Such transitions will play an important role when considering workflow equivalence. With these assumptions in mind, Figure 1 shows the semantics of basic workflow constructs.

For the semantics of the XOR-Split, consider Figure 2. Two alternative mappings are shown with the rightmost mapping commonly used in the literature (see e.g. [AAH98, SH96]). The semantics of the XOR-Split is that when completing activity A, a choice needs to be made for activity B or activity C. However, only one of them can be in the state `READY`. Hence, the rightmost mapping is incorrect, as after completion of activity A both activities B and C would be enabled (though still only one of them will be actually executed). This is not what can be observed for the majority of workflow systems - either B or C is enabled (appears on the worklist) but not both. The rightmost mapping would correspond to the Deferred Choice

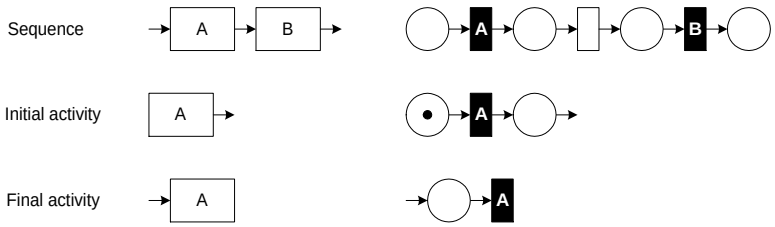

Figure 1: Mapping of basic control flow constructs

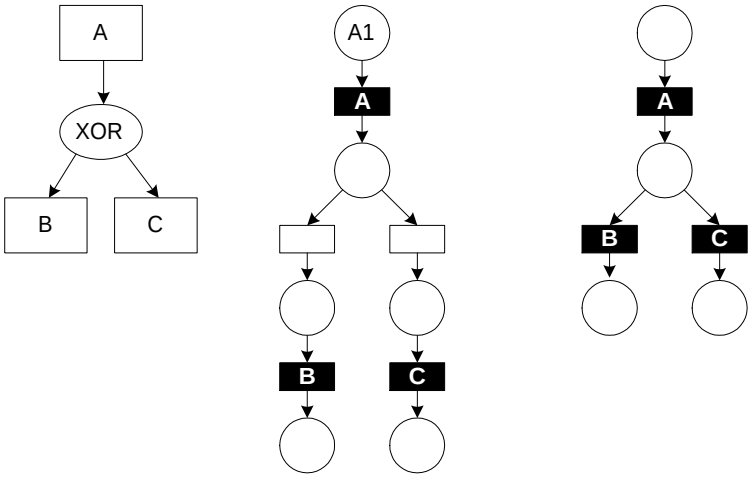

Figure 2: Alternative mappings for the XOR-Split

pattern, introduced in [ABHK00, AHKB00, AHKB02], and its importance will be immediate in later sections when discussing the expressiveness of Standard Workflow Models. Note that the semantics of the XOR-Split has been presented for the binary case, but, of course, can be trivially extended for the n-ary case (this will also hold for the other constructs presented in this paper).

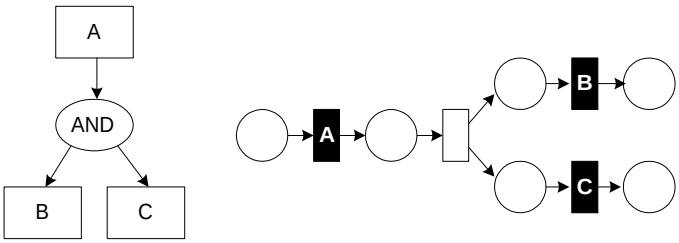

Figure 3: Mapping for the AND-Split

The interpretation of the AND-Split is presented in Figure 3, while interpretations for the AND-Join and the OR-Join are provided in Figure 4.

Note that the formal semantics provided for both types of joins leaves no ambiguities as to what is the semantics of these constructs when put in the context of a more complicated process structure. For example, if the AND-Join and activities A and B of Figure 4 were preceded by

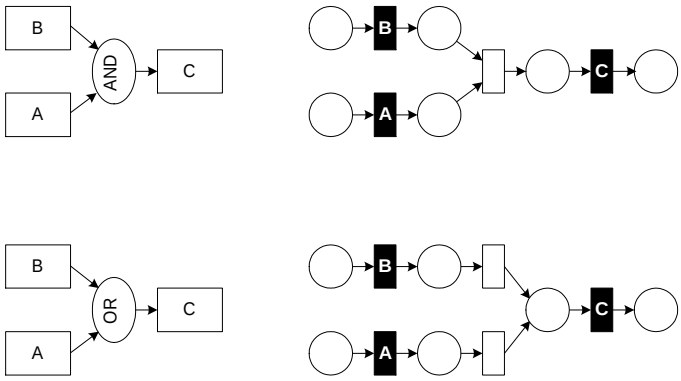

Figure 4: Mappings for the AND-Join and the OR-Join

a XOR-Split, only one incoming activity (say, activity B) could complete. In this case there would be a token in c_B (c_B is the output place of activity B, indicating the completion of B) and the subsequent transition will never fire as no token would ever reach c_A. (c_A is the output place of A.) The net would then be in deadlock. If, on the other hand, the OR-Join and activities A and B of Figure 4 were preceded by the AND-Split, both activities B and A could run in parallel and tokens would be produced for both c_B and c_A. As a result two tokens would end up in r_C (r_C is the input place of C, indicating that C is ready to be executed). (Note that these tokens are not necessarily at the same point in time in r_C.). This corresponds to a situation where activity C has to be performed twice (which may or may not be desirable). In a workflow context this behaviour is observable, as any user that has been assigned to perform activity C will see two instances of this activity on his/her worklist.

Having informally established what a Standard Workflow Model is and how its constructs should be mapped to Petri nets, the formal definition of such a net (Definition 2.1) and its mapping (Definition 2.3) can be presented.

Definition 2.1

A Standard Workflow Model *is a tuple* $\mathcal{W} = (\mathcal{P}, \mathcal{J}_o, \mathcal{J}_a, \mathcal{S}_o, \mathcal{S}_a, \mathcal{A}, \mathsf{Trans}, \mathsf{Name})$ *where* $\mathcal{P}$ *is a set of process elements which can be further divided into disjoint sets of OR-Joins* $\mathcal{J}_o$*, AND-Joins* $\mathcal{J}_a$*, XOR-Splits* $\mathcal{S}_o$*, AND-Splits* $\mathcal{S}_a$*, and activities* $\mathcal{A}$*;* $\mathsf{Trans} \subseteq \mathcal{P} \times \mathcal{P}$ *is a transition relation between process elements and* $\mathsf{Name} \in \mathcal{N}^{\mathcal{A}}$ *is a function assigning names to activities taken from some given set of names* $\mathcal{N}$ *containing special label* λ*.*

Activities without names[3] *are referred to as null activities. Joins have an outdegree of at most one, while splits have an indegree of at most one. Activities have an indegree and outdegree of at most one. Finally, we will call activities with an indegree of zero* initial items *(*$\mathcal{I} \subseteq \mathcal{A}$*) and all process elements with an outdegree of zero* final items *(*$\mathcal{F} \subseteq \mathcal{P}$*).*

□

If confusion is possible, we add superscripts to the elements of a Standard Workflow Model. For example, $\mathcal{A}^{\mathcal{W}}$ denotes the set of activities $\mathcal{A}$ of a Standard Workflow Model $\mathcal{W}$. Additionally we

[3] For an activity a without a name, we have $\mathsf{Name}(a) = \lambda$.

will use the notation $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ whenever there is no need to distinguish between the different types of process elements.

In Definition 2.1 we have imposed as few as possible syntactic restrictions. In this respect the following is worth noting:

- It may seem to be very restrictive to require that activities have an indegree and outdegree of at most one (and similar restrictions for the splits). This approach has been chosen to avoid possible ambiguities. For example, an activity with an indegree of two is sometimes interpreted as an AND-Join and sometimes as an OR-Join. It is trivial to map any language with such implicit semantics to our explicit notation.
- Most languages would require that the indegree of joins is at least one. Similarly they would require the outdegree of splits to be at least one. We have decided not to impose these restrictions as by not introducing these restrictions we can simplify our definition as well as some further proofs without loosing any generality.

Definition 2.2

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Standard Workflow Model and $e \in \mathcal{P}$ a process element of $\mathcal{W}$. Input elements of e are given by $in(e) = \{x \in \mathcal{P} \mid x \ \mathsf{Trans}\ e\}$ and output elements of e by $out(e) = \{x \in \mathcal{P} \mid e \ \mathsf{Trans}\ x\}$. □

Definition 2.3

Given a Standard Workflow Model $\mathcal{W} = (\mathcal{P}, \mathcal{J}_o, \mathcal{J}_a, \mathcal{S}_o, \mathcal{S}_a, \mathcal{A}, \mathsf{Trans}, \mathsf{Name})$, the corresponding labelled Petri net $PN_{\mathcal{W}} = (P_{\mathcal{W}}, T_{\mathcal{W}}, F_{\mathcal{W}}, L_{\mathcal{W}})$ is defined by:

$$
\begin{array}{lll}
P_{\mathcal{W}} = & \{r_{x,y} \mid x \in \mathcal{P} \wedge y \in in(x)\} \cup & \#\text{“ready” places}\# \\
& \{c_{x,y} \mid x \in \mathcal{P} \wedge y \in out(x)\} \cup & \#\text{“completed” places}\# \\
& \{r_x \mid x \in \mathcal{I}\} & \#\text{“initial” places}\# \\
 & & \\
T_{\mathcal{W}} = & \{X_{x,y} \mid x \in \mathcal{S}_o \wedge y \in out(x)\} \cup & \#\textit{XOR-Split}\# \\
& \{R_x \mid x \in \mathcal{S}_a\} \cup & \#\textit{AND-Split}\# \\
& \{K_x \mid x \in \mathcal{J}_a\} \cup & \#\textit{AND-Join}\# \\
& \{Q_{x,y} \mid x \in \mathcal{J}_o \wedge y \in in(x)\} \cup & \#\textit{OR-Join}\# \\
& \{A_x \mid x \in \mathcal{A}\} \cup & \#\textit{activity}\# \\
& \{L_{x,y} \mid x \ \mathsf{Trans}\ y\} & \#\textit{connecting trans.}\# \\
 & & \\
L_{\mathcal{W}} = & \{(A_x, \mathsf{Name}(x)) \mid x \in \mathcal{A}\} \cup & \#\textit{activities}\# \\
& \{(t, \lambda) \mid t \in T_{\mathcal{W}} \wedge \neg\exists_{x \in A}\,[t = A_x]\} & \#\textit{other trans}\# \\
 & & \\
F_{\mathcal{W}} = & \{(r_x, A_x) \mid x \in \mathcal{I}\} \cup & \#\textit{initial places}\# \\
& \{(r_{x,y}, A_x) \mid x \in \mathcal{A} \wedge y \in in(x)\} \cup & \\
& \{(A_x, c_{x,y}) \mid x \in \mathcal{A} \wedge y \in out(x)\} \cup & \#\textit{activity}\# \\
& \{(r_{x,y}, K_x) \mid x \in \mathcal{J}_a \wedge y \in in(x)\} \cup & \\
& \{(K_x, c_{x,y}) \mid x \in \mathcal{J}_a \wedge y \in out(x)\} \cup & \#\textit{AND-Join}\# \\
& \{(r_{x,y}, R_x) \mid x \in \mathcal{S}_a \wedge y \in in(x)\} \cup &
\end{array}
$$

$$
\begin{array}{ll}
\{(R_x, c_{x,y}) \mid x \in \mathcal{S}_a \wedge y \in out(x)\} \cup & \#AND\text{-}Split\# \\
\{(r_{x,y}, Q_{x,y}) \mid x \in \mathcal{J}_o \wedge y \in in(x)\} \cup & \\
\{(Q_{x,z}, c_{x,y}) \mid x \in \mathcal{J}_o \wedge y \in out(x) \wedge z \in in(x)\} \cup & \#OR\text{-}Join\# \\
\{(r_{x,y}, X_{x,z}) \mid x \in \mathcal{S}_o \wedge y \in in(x) \wedge z \in out(x)\} \cup & \\
\{(X_{x,y}, c_{x,y}) \mid x \in \mathcal{S}_o \wedge y \in out(x)\} \cup & \#XOR\text{-}Split\# \\
\{(c_{x,y}, L_{x,y}) \mid x \ \mathsf{Trans} \ y\} \cup & \\
\{(L_{x,y}, r_{y,x}) \mid x \ \mathsf{Trans} \ y\} & \#connecting\#
\end{array}
$$

□

Definition 2.4

Given a Standard Workflow Model $\mathcal{W}$, the corresponding net system of $\mathcal{W}$ is a pair $(PN_{\mathcal{W}}, M_0)$ where $PN_{\mathcal{W}}$ is the corresponding net and M_0 is an initial marking that assigns a single token to each of the places in $\{r_x \mid x \in \mathcal{I}\}$. □

We will often refer to Petri nets resulting from the translation of Standard Workflow Models as *Standard Workflow Nets.*

Though the definition of a Standard Workflow Model may look complicated, it is constructed from a number of elementary building blocks, which can be isolated through Definition 2.5 if required.

Definition 2.5

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Standard Workflow Model and $PN_{\mathcal{W}} = (P_{\mathcal{W}}, T_{\mathcal{W}}, F_{\mathcal{W}}, L_{\mathcal{W}})$ its corresponding net. The associated net of a process element $e \in \mathcal{P}$, $PN^e_{\mathcal{W}} = (P^e_{\mathcal{W}}, T^e_{\mathcal{W}}, F^e_{\mathcal{W}}, L^e_{\mathcal{W}})$, is a subnet of $PN_{\mathcal{W}}$ and is defined by:

$$
\begin{array}{rcll}
P^e_{\mathcal{W}} & = & \begin{cases} \{r_{e,i} \mid i \in in(e)\} \cup \{c_{e,o} \mid o \in out(e)\} & \text{if } e \notin \mathcal{I} \\ \{r_e\} \cup \{c_{e,o} \mid o \in out(e)\} & \text{if } e \in \mathcal{I} \end{cases} \\
T^e_{\mathcal{W}} & = & \{t \in T_{\mathcal{W}} \mid {\bullet t} \subseteq P^e_{\mathcal{W}} \wedge t{\bullet} \subseteq P^e_{\mathcal{W}}\} & \\
F^e_{\mathcal{W}} & = & F_{\mathcal{W}} \cap (P^e_{\mathcal{W}} \times T^e_{\mathcal{W}} \cup T^e_{\mathcal{W}} \times P^e_{\mathcal{W}}) & \\
L^e_{\mathcal{W}} & = & L_{\mathcal{W}}[T^e_{\mathcal{W}}] &
\end{array}
$$

□

Example 2.1 As an example of the application of Definitions 2.3 and 2.5 consider the Standard Workflow Model and its corresponding mapping along with associated nets in Figure 5.

□

Having formally defined Standard Workflow Models, it is now possible to precisely define properties of such models, which have been informally referred to earlier in this section. First it is useful to be able to talk about running instances of workflows.

Definition 2.6

An instance *of a Standard Workflow Model $\mathcal{W}$ is a marking reachable from the initial marking of its corresponding net system.* □

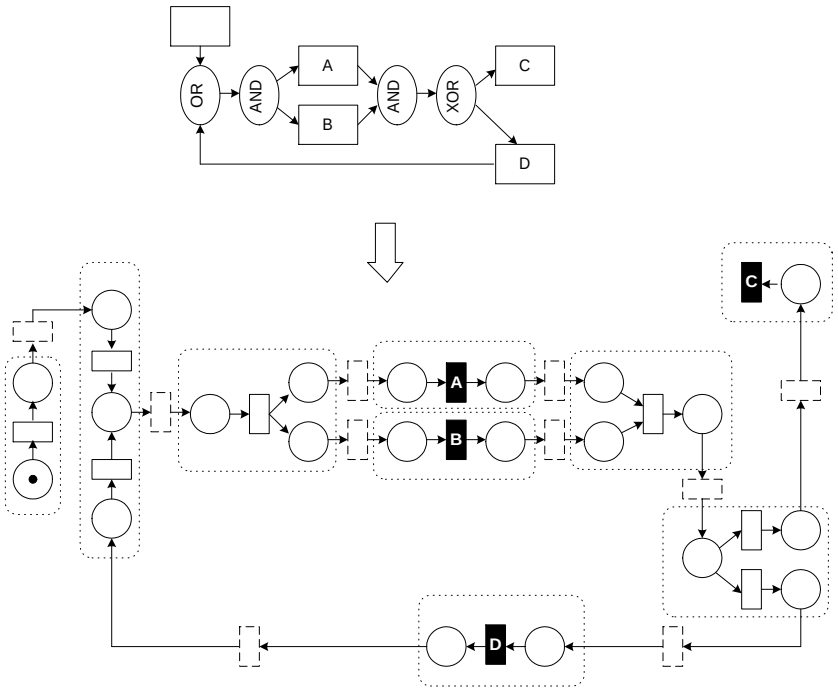

Figure 5: Sample Standard Workflow Model and its corresponding Petri net

The next definition formally defines what it means to *enable* a process element and *fire* a process element.

Definition 2.7

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ *be a Standard Workflow Model and* $e \in \mathcal{P}$ *a process element of* $\mathcal{W}$*. We say that* e *is* enabled *in an instance* M *of* $\mathcal{W}$ *if any transition of its associated net* $PN^e_{\mathcal{W}}$ *is enabled in marking* M*. Similarly* firing *a process element* e *means firing any transition of its associated net.* □

Execution of a (finite) Standard Workflow Model leads either to a successful termination or to a deadlock or to an infinite loop from which the empty marking cannot be reached. More formally:

Definition 2.8

An instance of a Standard Workflow Model $\mathcal{W}$ *is in* deadlock *iff it is not the empty marking and no transition is enabled.* □

Definition 2.9

An instance of a Standard Workflow Model $\mathcal{W}$ *is in an* infinite loop *iff it is not the empty marking and there is no firing sequence that leads to either the empty marking or to a deadlock.* □

Definition 2.10

A Standard Workflow Model $\mathcal{W}$ is deadlock-free *iff none of its instances is in a deadlock.* □

Definition 2.11

A Standard Workflow Model $\mathcal{W}$ is terminating *iff from all its instances the empty marking can be reached.* □

Terminating Standard Workflow Models are similar to the class of WorkFlow Nets (WF-nets, cf. [Aal98]). One of the differences is that WF-nets have an explicit termination place, i.e., a place which once it gets marked corresponds to a terminated workflow.

In previous publications [AHKB00, AHKB02] we have referred to the term *multiple instances* as the situation in which one activity may have many instances of it running concurrently. Based on our definition of the semantics of Standard Workflow Nets, we can provide a more formal definition.

Definition 2.12

A Standard Workflow Model does not have multiple instances *iff for every place p of its corresponding net system and for every reachable marking M, $M(p) \leq 1$. We will call such models* safe. □

At a first glance is may appear strange that multiple instances and non-safeness coincide. However, if each input place of a transition contains multiple tokens, then this transition is concurrently enabled with itself. Therefore, it is natural to relate multiple tokens in a place (non-safe) to multiple instances as defined in [AHKB00, AHKB02].

Definition 2.13

A Standard Workflow Model is bounded *iff the corresponding net system is bounded (i.e., there is only a finite number of reachable markings).* □

Finally we will refer to Standard Workflows that are safe and terminating as *well-behaved.*

Definition 2.14

A Standard Workflow Model $\mathcal{W}$ is well-behaved *iff it is safe and it is terminating.* □

2.2 Safe Workflow Models

The main difference between *Safe Workflow Models* and Standard Workflow Models is the behaviour of the OR-Join. As the WfMC does not define what should happen if more than one thread input to the OR-Join is concurrently active, some workflow management systems (e.g. Staffware, HP Changengine, Fujitsu's i-Flow) have been based on the assumption that subsequent active threads should never reach the OR-Join. Hence, their engines will never

create multiple concurrent instances of the activity following the OR-Join. Though the actual solution is different for different products, from a conceptual point of view, the result is the same: there is no direct support for multiple instances.

To formally characterise such languages, two approaches could have been taken. One way would be to define the Petri net semantics of activities such that an activity's `READY` place can never hold more than one token. In that case it is guaranteed that the corresponding Petri net will be safe. This approach would try to formalize the *observable* behaviour of languages such as Staffware.

Following our discussions with vendors who have chosen the safe evaluation strategy, we have decided to take a different approach. It is based on the assumption that processes resulting in multiple active threads input to an OR-Join are considered to be flawed and their semantics is undefined.

This allows us to simply view Safe Workflow Models as a subclass of Standard Workflow Models.

Definition 2.15

A Safe Workflow Model *is a Standard Workflow Model such that its corresponding net system is safe.* □

2.3 Synchronizing Workflow Models

Synchronizing Workflow Models form a third class of workflow languages based on yet another, fundamentally different, interpretation of the WfMC definitions of the basic control flow constructs. The intuitive reasoning here is as follows. An AND-Join typically follows an AND-Split and can be seen as a construct that synchronizes a number of active threads. An OR-Join on the other hand, typically follows an exclusive XOR-Split. While there is only one active thread of execution in that case, the OR-Join can still be seen as a construct that synchronizes threads: one active and the others inactive. The active thread propagates a "True" token, while an inactive thread propagates a "False" token. Hence both types of joins synchronize a number of threads of execution. With slight modifications, this view was successfully implemented by IBM's MQSeries Workflow (formerly known as FlowMark) [LR99]. Note that a synchronizing strategy prevents the use of arbitrary cycles [ABHK00, AHKB00, AHKB02, WPH02] (as that would immediately lead to a deadlock). Later it will be proven that Synchronizing Workflow Models never deadlock.

The problem that an OR-join may need to synchronize depending on the number of active threads was also investigated in the context of Event-driven Process Chains (EPC's, cf. [KNS92]). EPC's allow for so-called $\vee$-connectors (i.e., OR-joins which only synchronize the flows that are active). The semantics of these $\vee$-connectors have often been debated [Aal99, DR01, LSW98, Rit99, Rum97]. By using a Synchronizing Workflow Model it is possible to give a precise and intuitive semantics.

The semantics of Synchronizing Workflow Models is very naturally captured using Coloured Petri nets [Jen87] or Predicate/Transition nets [Gen87], as they allow for typed tokens with identity. There are also relations with the so-called "Bipolar Synchronization Schemes" introduced in [GT84] where tokens have an "H" or "L" value. However, in order to facilitate a

formal comparison with Standard Workflow Models, we provide a formal semantics in terms of standard Petri nets.

In Synchronizing Workflow Models, an activity can receive two types of tokens, a true token or a false token. Receipt of a true token should enable the activity, while receipt of a false token should lead to the activity being skipped and the token to be propagated. To capture this in standard Petri nets, we divide the set of places into places that capture the receipt of true tokens ("true places") and places that capture the receipt of false tokens ("false places"). This leads to the semantics represented in Figure 6.

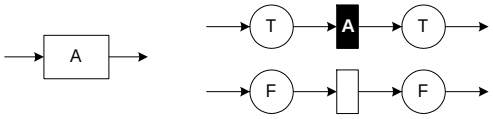

Figure 6: Activity semantics for Synchronizing Workflow Models

Each activity x has a place $rt_{x,y}$ in its corresponding Petri net (where y corresponds to the input element of x, if existing) that will hold a token if the activity received a true token from y, and a place $rf_{x,y}$ that will hold a token if the activity received a false token from y. As is clear from the net in Figure 6, a token in a "true" place will lead to the transition labelled with A being enabled, while a token in its "false" place will lead to the non-labelled transition being enabled, and hence nothing, other than propagation of the token, will happen.

The semantics of the XOR-Split and the AND-Split is relatively straightforward. When a true token arrives, a XOR-Split will pass on a true token to one of its outgoing branches and false tokens for all the other outgoing branches. When a true token arrives for an AND-Split, true tokens are passed on to all its outgoing branches. Both splits behave similar when receiving a false token; it is simply passed on to all outgoing branches. This semantics is captured in Figure 7.[4]

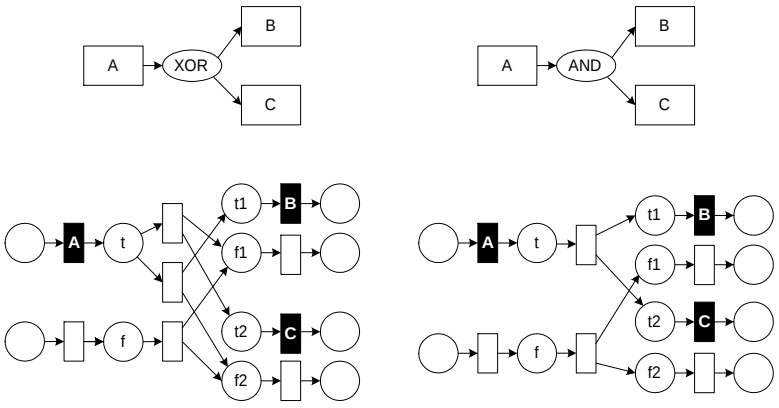

Figure 7: Split semantics for Synchronizing Workflow Models

More interesting is the semantics of the join constructs. As noted earlier, in Synchronizing

[4]Note that the connecting L-transitions (i.e., LT and LF) have been omitted for reasons of simplicity (cf. Definition 2.18).

Workflow Models a join construct always waits for a token to arrive from every incoming transition. The only differentiator between different types of joins could be the type of tokens expected. In this paper we will follow MQSeries/Workflow in that we will distinguish two cases - an *ANY-Join* which passes on a true token if it received at least one true token (otherwise it passes on a false token) and the *ALL-Join* which passes on a true token if it received true tokens from all incoming branches (otherwise it passes on a false token). Later, in Section 4.3, we will show how the Synchronizing Workflow's *ANY-Join* and *ALL-Join* correspond to the Standard and Safe Workflow's *OR-Join* and *AND-Join*.

In Figure 8, the semantics of the joins is shown in the context of Synchronizing Workflow Models.

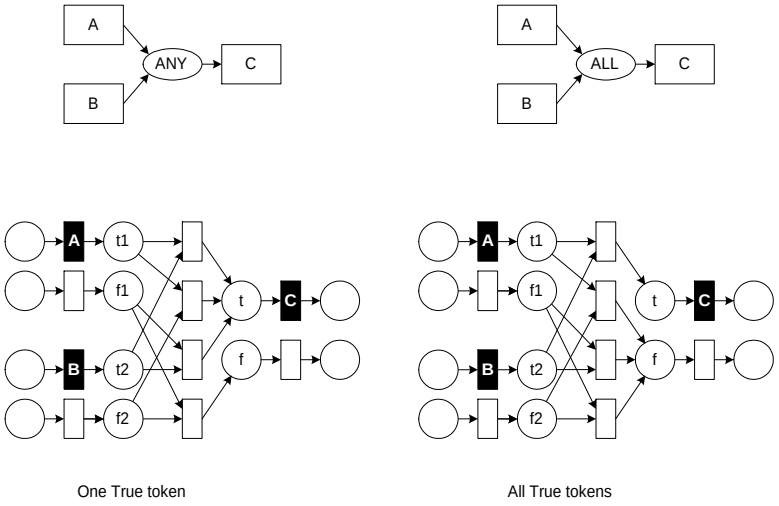

Figure 8: Join semantics for Synchronizing Workflow Models

A free-choice Petri net is a net in which the choice between two transitions competing for the same token is never influenced by the rest of the system [DE95]. On a structural level it means that a Petri net is free-choice iff for every place p and every transition t of this net: if there is an arc from a p to t, then there must be an arc from any input place of t to any output transition of p (see Appendix A for a formal definition of a free-choice Petri net). One important characterisation of Synchronizing Workflow Models is that the Petri net representation of the join constructs is not free-choice (see [DE95] for a detailed discussion of free-choice Petri nets). In Section 4.3 it will be shown that some Synchronizing Workflow Models are inherently non free-choice.

Having informally established the semantics of Synchronizing Workflow Models, Definition 2.16 formally defines their syntax, while Definition 2.18 formally defines their semantics.

Definition 2.16

> *A* Synchronizing Workflow Model *is a tuple* $\mathcal{W} = (\mathcal{P}, \mathcal{J}_o, \mathcal{J}_a, \mathcal{S}_o, \mathcal{S}_a, \mathcal{A}, \mathsf{Trans}, \mathsf{Name})$ *where* $\mathcal{P}$ *is a set of process elements which can be further divided into disjoint sets of ANY-Joins* $\mathcal{J}_o$*, ALL-Joins* $\mathcal{J}_a$*, XOR-Splits* $\mathcal{S}_o$*, AND-Splits* $\mathcal{S}_a$*, and activities* $\mathcal{A}$*;*

$\mathsf{Trans} \subseteq \mathcal{P} \times \mathcal{P}$ *is a transition relation between process elements and* $\mathsf{Name} \in \mathcal{N}^{\mathcal{A}}$ *is a function assigning names to activities taken from some given set of names* $\mathcal{N}$ *containing special label* λ.

Activities without names are referred to as null activities. Joins have an indegree of at least one and an outdegree of one, while splits have an indegree of one and an outdegree of at least one. Activities have an indegree and outdegree of at most one. Finally, we will call activities with an indegree of zero initial items *(*$\mathcal{I} \subseteq \mathcal{A}$*) and conversely, activities with an outdegree of zero* final items *(*$\mathcal{F} \subseteq \mathcal{A}$*).* □

Note that the syntax of Synchronizing Workflow Models is very similar to the syntax of Standard Workflow Models. The only difference is that joins and splits cannot have indegree or outdegree of zero (this is to allow simplification of the semantics).

The following definition provides auxiliary functions and predicates that facilitate the specification of the formal semantics.

Definition 2.17

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ *be a Synchronizing Workflow Model and* $p \in \mathcal{P}$ *a process element. The input elements of* p *are given by* $in(p) = \{x \in \mathcal{P} \mid x \ \mathsf{Trans} \ p\}$ *and output elements of* p *by* $out(p) = \{x \in \mathcal{P} \mid p \ \mathsf{Trans} \ x\}$. *Further, if* $b \in \{t, f\}^A$ *is a function with domain* A *(which is nonempty), then* $alltrue(b)$ *holds iff* $\forall_{a \in A}\,[b(a) = t]$ *and* $allfalse(b)$ *holds iff* $\forall_{a \in A}\,[b(a) = f]$. □

Definition 2.18

Given a Synchronising Workflow Model $\mathcal{W}$*, the corresponding labelled Petri net* $PN_{\mathcal{W}} = (P_{\mathcal{W}}, T_{\mathcal{W}}, F_{\mathcal{W}}, L_{\mathcal{W}})$ *is defined by:*

$$
\begin{array}{lll}
P_{\mathcal{W}} = & \{rt_{x,i} \mid x \in \mathcal{P} \wedge i \in in(x)\} \cup & \#\text{“ready” } true\# \\
& \{rf_{x,i} \mid x \in \mathcal{P} \wedge i \in in(x)\} \cup & \#\text{“ready” } false\# \\
& \{ct_{x,o} \mid x \in \mathcal{P} \wedge o \in out(x)\} \cup & \#\text{“completed” } true\# \\
& \{cf_{x,o} \mid x \in \mathcal{P} \wedge o \in out(x)\} \cup & \#\text{“completed” } false\# \\
& \{rt_x \mid x \in \mathcal{I}\} \cup & \#\text{“initial” } true\# \\
& \{rf_x \mid x \in \mathcal{I}\} & \#\text{“initial” } false\# \\
 & & \\
T_{\mathcal{W}} = & \{XT_{x,o} \mid x \in \mathcal{S}_o \wedge o \in out(x)\} \cup \{XF_x \mid x \in \mathcal{S}_o\} \cup & \#XOR\text{-}Split\# \\
& \{RF_x \mid x \in \mathcal{S}_a\} \cup \{RT_x \mid x \in \mathcal{S}_a\} \cup & \#AND\text{-}Split\# \\
& \{K_x^b \mid x \in \mathcal{J}_a \wedge b \in \{t, f\}^{in(x)}\} \cup & \#ALL\text{-}Join\# \\
& \{Q_x^b \mid x \in \mathcal{J}_o \wedge b \in \{t, f\}^{in(x)}\} \cup & \#ANY\text{-}Join\# \\
& \{AF_x \mid x \in \mathcal{A}\} \cup \{AT_x \mid x \in \mathcal{A}\} \cup & \#activity\# \\
& \{LT_{x,y} \mid x \ \mathsf{Trans} \ y\} \cup \{LF_{x,y} \mid x \ \mathsf{Trans} \ y\} & \#connecting\ trans.\# \\
 & & \\
L_{\mathcal{W}} = & \{(AT_x, \mathsf{Name}(x)) \mid x \in \mathcal{A}\} \cup & \#activities\# \\
& \{(t, \lambda) \mid t \in T_{\mathcal{W}} \wedge \neg\exists_{x \in A}\,[t = AT_x]\} & \#other\ trans\# \\
 & & \\
F_{\mathcal{W}} = & \{(rt_x, AT_x) \mid x \in \mathcal{I}\} \cup &
\end{array}
$$

$$
\begin{array}{ll}
\{(rf_x, AF_x) \mid x \in \mathcal{I}\} \cup & \#\textit{initial places}\# \\
\{(rt_{x,i}, AT_x) \mid x \in \mathcal{A} \wedge i \in in(x)\} \cup & \\
\{(AT_x, ct_{x,o}) \mid x \in \mathcal{A} \wedge o \in out(x)\} \cup & \\
\{(rf_{x,i}, AF_x) \mid x \in \mathcal{A} \wedge i \in in(x)\} \cup & \\
\{(AF_x, cf_{x,o}) \mid x \in \mathcal{A} \wedge o \in out(x)\} \cup & \#\textit{activity}\# \\
\{(rt_{x,i}, RT_x) \mid x \in \mathcal{S}_a \wedge i \in in(x)\} \cup & \\
\{(RT_x, ct_{x,o}) \mid x \in \mathcal{S}_a \wedge o \in out(x)\} \cup & \\
\{(rf_{x,i}, RF_x) \mid x \in \mathcal{S}_a \wedge i \in in(x)\} \cup & \\
\{(RF_x, cf_{x,o}) \mid x \in \mathcal{S}_a \wedge o \in out(x)\} \cup & \#\textit{AND-Split}\# \\
\{(rf_{x,i}, XF_x) \mid x \in \mathcal{S}_o \wedge i \in in(x)\} \cup & \\
\{(XF_x, cf_{x,o}) \mid x \in \mathcal{S}_o \wedge o \in out(x)\} \cup & \\
\{(rt_{x,i}, XT_{x,o}) \mid x \in \mathcal{S}_o \wedge i \in in(x) \wedge o \in out(x)\} \cup & \\
\{(XT_{x,o1}, cf_{x,o2}) \mid x \in \mathcal{S}_o \wedge \{o1, o2\} \subseteq out(x) \wedge o1 \neq o2\} \cup & \\
\{(XT_{x,o1}, ct_{x,o1}) \mid x \in \mathcal{S}_o \wedge o1 \in out(x)\} \cup & \#\textit{XOR-Split}\# \\
\{(rt_{x,i}, K^b_x) \mid x \in \mathcal{J}_a \wedge i \in in(x) \wedge b \in \{t,f\}^{in(x)} \wedge b(i) = t\} \cup & \\
\{(K^b_x, ct_{x,o}) \mid x \in \mathcal{J}_a \wedge o \in out(x) \wedge b \in \{t,f\}^{in(x)} \wedge \textit{alltrue}(b)\} \cup & \\
\{(rf_{x,i}, K^b_x) \mid x \in \mathcal{J}_a \wedge i \in in(x) \wedge b \in \{t,f\}^{in(x)} \wedge b(i) = f\} \cup & \\
\{(K^b_x, cf_{x,o}) \mid x \in \mathcal{J}_a \wedge o \in out(x) \wedge b \in \{t,f\}^{in(x)} \wedge \neg\textit{alltrue}(b)\} \cup & \\
 & \#\textit{ALL-Join}\# \\
\{(rt_{x,i}, Q^b_x) \mid x \in \mathcal{J}_o \wedge i \in in(x) \wedge b \in \{t,f\}^{in(x)} \wedge b(i) = t\} \cup & \\
\{(Q^b_x, ct_{x,o}) \mid x \in \mathcal{J}_o \wedge o \in out(x) \wedge b \in \{t,f\}^{in(x)} \wedge \neg\textit{allfalse}(b)\} \cup & \\
\{(rf_{x,i}, Q^b_x) \mid x \in \mathcal{J}_o \wedge i \in in(x) \wedge b \in \{t,f\}^{in(x)} \wedge b(i) = f\} \cup & \\
\{(Q^b_x, cf_{x,o}) \mid x \in \mathcal{J}_o \wedge o \in out(x) \wedge b \in \{t,f\}^{in(x)} \wedge \textit{allfalse}(b)\} \cup & \\
 & \#\textit{ANY-Join}\# \\
\{(ct_{x,y}, LT_{x,y}) \mid x \ \mathsf{Trans}\ y\} \cup & \\
\{(LT_{x,y}, rt_{y,x}) \mid x \ \mathsf{Trans}\ y\} \cup & \\
\{(cf_{x,y}, LF_{x,y}) \mid x \ \mathsf{Trans}\ y\} \cup & \\
\{(LF_{x,y}, rf_{y,x}) \mid x \ \mathsf{Trans}\ y\} & \#\textit{connecting ready/completed}\#
\end{array}
$$

□

Definition 2.19

Given a Synchronizing Workflow Model $\mathcal{W}$, the corresponding net system of $\mathcal{W}$ is a pair $(PN_{\mathcal{W}}, M_0)$ where $PN_{\mathcal{W}}$ is the corresponding net of $\mathcal{W}$ and M_0 is an initial marking that assigns a single token to each of the places in $\{rt_x \mid x \in \mathcal{I}\}$. □

We will often refer to Petri nets resulting from the translation of Synchronizing Workflow Models as *Synchronizing Workflow Nets*.

Example 2.2 As an example of the application of Definition 2.18 consider the Synchronizing Workflow Model and its corresponding mapping in Figure 9. □

Similarly to Standard Workflow Models, Synchronizing Workflow Models are constructed from a number of elementary building blocks, which can be isolated through Definition 2.20.

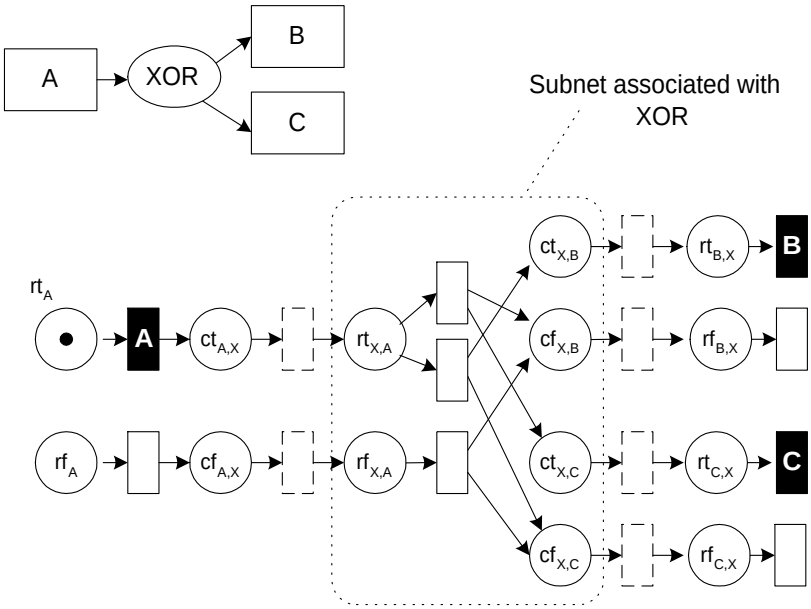

Figure 9: Synchronizing Workflow Model and its corresponding Petri net

Definition 2.20

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ *be a Synchronizing Workflow Model and* $PN_{\mathcal{W}} = (P_{\mathcal{W}}, T_{\mathcal{W}}, F_{\mathcal{W}}, L_{\mathcal{W}})$ *its corresponding Petri net. Let* $e \in \mathcal{P}_{\mathcal{W}}$ *be a process element. The associated net,* $PN^e_{\mathcal{W}} = (P^e_{\mathcal{W}}, T^e_{\mathcal{W}}, F^e_{\mathcal{W}}, L^e_{\mathcal{W}})$*, a subnet of* $PN_{\mathcal{W}}$*, is defined by:*

$$
\begin{array}{lcll}
P^e_{\mathcal{W}} & = & \left\{ \begin{array}{ll} \{rt_{e,i} \mid i \in in(e)\} \cup \{rf_{e,i} \mid i \in in(e)\} \cup & \\ \{ct_{e,o} \mid o \in out(e)\} \cup \{cf_{e,o} \mid o \in out(e)\} & \text{if } e \notin \mathcal{I} \\ \{rt_e, rf_e\} \cup \{ct_{e,o} \mid o \in out(e)\} \cup \{cf_{e,o} \mid o \in out(e)\} & \text{if } e \in \mathcal{I} \end{array} \right. \\
T^e_{\mathcal{W}} & = & \{t \in T_{\mathcal{W}} \mid \bullet t \subseteq P^e_{\mathcal{W}} \wedge t\bullet \subseteq P^e_{\mathcal{W}}\} & \\
F^e_{\mathcal{W}} & = & F_{\mathcal{W}} \cap (P^e_{\mathcal{W}} \times T^e_{\mathcal{W}} \cup T^e_{\mathcal{W}} \times P^e_{\mathcal{W}}) & \\
L^e_{\mathcal{W}} & = & L_{\mathcal{W}}[T^e_{\mathcal{W}}] &
\end{array}
$$

□

Example 2.3 The associated net for the XOR-Split is illustrated in Figure 9. □

Synchronizing Workflow Models have a more complicated Petri net translation because each process element can receive a "true" or a "false" token, and for that reason we introduce two input and two output places for each incoming and outgoing transition respectively. This is captured formally in the following definition.

Definition 2.21

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ *be a Synchronizing Workflow Model and* $PN_{\mathcal{W}}$ *its corresponding Petri net. The set of its* true places *is defined by*

$$\mathsf{True}^{\mathcal{W}} = \{rt_{x,i} \mid x \in \mathcal{P} \wedge i \in in(x)\} \cup \{ct_{x,o} \mid x \in \mathcal{P} \wedge o \in out(x)\} \cup \{rt_i \mid i \in \mathcal{I}\},$$

while the set of its false places *is given by:*

$$\mathsf{False}^{\mathcal{W}} = \{rf_{x,i} \mid x \in \mathcal{P} \wedge i \in in(x)\} \cup \{cf_{x,o} \mid x \in \mathcal{P} \wedge o \in out(x)\} \cup \{rf_i \mid i \in \mathcal{I}\}.$$

□

In an informal discussion earlier in this section we have often referred to the propagation of a "true" token or a "false" token. Formally we will call any token in a "true" place a "true" token and any token in a "false" place a "false" token.

Each incoming and outgoing transition of a process element has exactly one "true" place and one "false" place. The following definition captures the relationship between true and false places of the same workflow construct:

Definition 2.22

Let $\mathcal{W}$ be a Synchronizing Workflow Model and $PN_{\mathcal{W}}$ its corresponding net. If p is a true place in the net $PN_{\mathcal{W}}$, then its corresponding false place $\overline{p}$ is $rf_{x,y}$ if $p = rt_{x,y}$, rf_x if $p = rt_x$ and it is $cf_{x,y}$ if $p = ct_{x,y}$. Similarly, $\overline{p}$ will yield the corresponding true place if p is a false place. □

The definition of a workflow instance, deadlock and termination for Synchronizing Workflow Models are analogous to that of Standard Workflow Models. However, given the different Petri net translation the notion of a process element being *enabled* is slightly different and informally it means that for each incoming branch exactly one of the two (true or false) corresponding input places holds a token.

Definition 2.23

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Synchronizing Workflow Model. A process element $e \in \mathcal{P}$ is enabled *in a marking M of its associated net $PN^e_{\mathcal{W}}$ iff for all x such that $x \in in(e)$*

$$(M(rt_{e,x}) = 1 \wedge M(rf_{e,x}) = 0) \vee (M(rt_{e,x}) = 0 \wedge M(rf_{e,x}) = 1),$$

and for all y such that $y \in out(e)$

$$M(ct_{e,y}) = 0 \wedge M(cf_{e,y}) = 0.$$

□

In the context of Synchronizing Workflow Models it is also useful to talk about a process element being completed, which then means that for each outgoing branch exactly one of the two (true or false) corresponding output places holds a token.

Definition 2.24

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Synchronizing Workflow Model. A process element $e \in \mathcal{P}$ is completed *in a marking M of its associated net $PN^e_{\mathcal{W}}$ iff for all x such that $x \in in(e)$*

$$M(rt_{e,x}) = 0 \wedge M(rf_{e,x}) = 0,$$

and for all y such that $y \in out(e)$

$$(M(ct_{e,y}) = 0 \wedge M(cf_{e,y}) = 1) \vee (M(ct_{e,y}) = 1 \wedge M(cf_{e,y}) = 0).$$

□

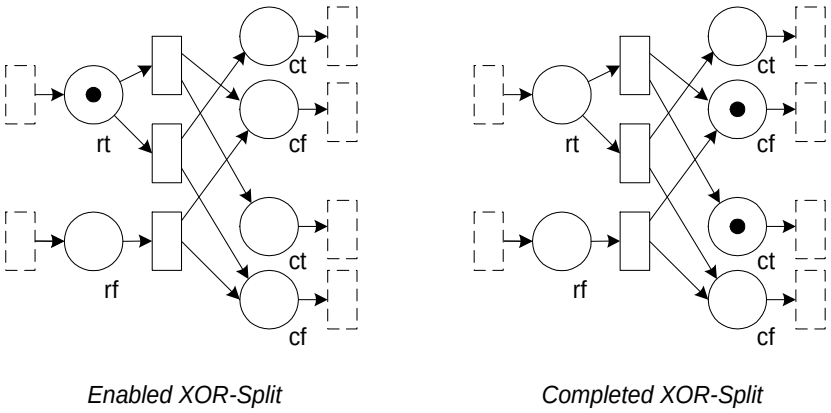

Figure 10: Enabled and completed XOR-Split

Example 2.4 Figure 10 shows markings in which an XOR-Split is *enabled* and *completed*. □

Finally, the following definition defines what it means to fire a process element.

Definition 2.25

Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ *be a Synchronizing Workflow Model and* $e \in \mathcal{P}$ *a process element which is enabled in marking* M *of its associated net* $PN^e_{\mathcal{W}}$. Firing e *means firing an enabled transition* t *of* $PN^e_{\mathcal{W}}$. □

The careful reader may notice that the definition of enabled process element for a Synchronizing Workflow Model is more restrictive than the similar definition for a Standard Workflow Model. This is due to the fact that the execution model for both workflows is fundamentally different. This issue will be further explored in Section 4.3.

3 Equivalence in the context of control flow

Sometimes workflow designers are faced with the task of transforming workflow specifications, for example to meet the particular requirements of a specific workflow engine. Naturally, such transformations should not alter the semantics of the original workflow, and as such they should be *equivalence preserving*. Similarly, when assessing the expressive power of a given workflow language the issue of equivalence is crucial. If one would like to prove that for a certain workflow a corresponding workflow in another language does, or does not, exist, this all depends on the notion of equivalence chosen.

For processes many different equivalence notions exist (e.g. trace, readiness, possible futures, fully concurrent bisimulation etc.) [Mil80, Mil89, Mil99]. In fact, a whole area of research is devoted to this topic, referred to as "comparative concurrency semantics" (for an overview of many equivalence notions, refer to e.g. [Gla90, Gla93, PRS92]).

In the context of workflows, the choice of the "right" notion of equivalence is very much an open issue. The equivalence notion chosen should not be too restrictive as that would mean that workflows that one would like to consider as behaving identically, would be considered to be fundamentally different. Similarly, an equivalence notion should not be too relaxed, as it would identify workflows that behave fundamentally differently. Naturally this issue, to some extent, is open for debate as it depends on intuition as regards workflow execution and on what one considers to be a "workable" enough definition.

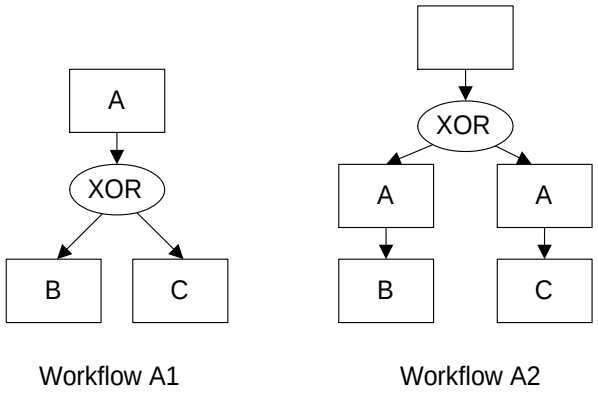

Figure 11: Two trace equivalent processes

Consider the workflows $A1$ and $A2$ in Figure 11. These workflows produce identical traces, namely *ab* and *ac*. In other words they are *trace equivalent*. From a practical point of view, however, one would not like to consider them to be equivalent, as the moment of choice in both workflows is different. The choice for activity B or activity C may be influenced by the data produced by activity A in the left workflow, but not in the right. Clearly *trace equivalence* is not strong enough to distinguish these two workflows. Equivalence notions that take into account decision points are typically referred to as equivalence notions preserving *branching time* (as opposed to linear time) [Gla90]. There are many equivalence notions that satisfy this criterion.

Considering only equivalence notions preserving branching time, we face a choice between *interleaving* semantics and the more complex *concurrent* semantics. In interleaving semantics, a process consisting of two tasks, A and B which run in parallel is equivalent to a process that chooses between running sequentially A followed by B or B followed by A. In other words, there is no *true concurrency* [PRS92]. As an example consider the two Petri nets, PN_1 and PN_2 of Figure 12. These two nets are equivalent under any interleaving equivalence notion. However we would like to consider workflows $B1$ and $B2$ of this figure to be semantically different. The standard way of dealing with this problem (see e.g. [BW90]) is to split a task into two observable transitions. Firing the first transition indicates *starting* of the task and firing the second transition indicates *completion* of the task. Having two parallel tasks A and B it is possible to obtain a trace $A_S B_S A_F B_F$ where A_S and B_S indicate start of tasks A and B respectively and A_F and B_F completion of tasks A and B respectively. This mapping is shown in Figure 12. Clearly workflows $B1$ and $B2$, given the presented mapping to Petri nets, are not even trace equivalent. As the mappings to Petri nets, as presented in Sections 2.1 and 2.3 map tasks to a subnet containing one labelled transition, for two workflows to be equivalent we will

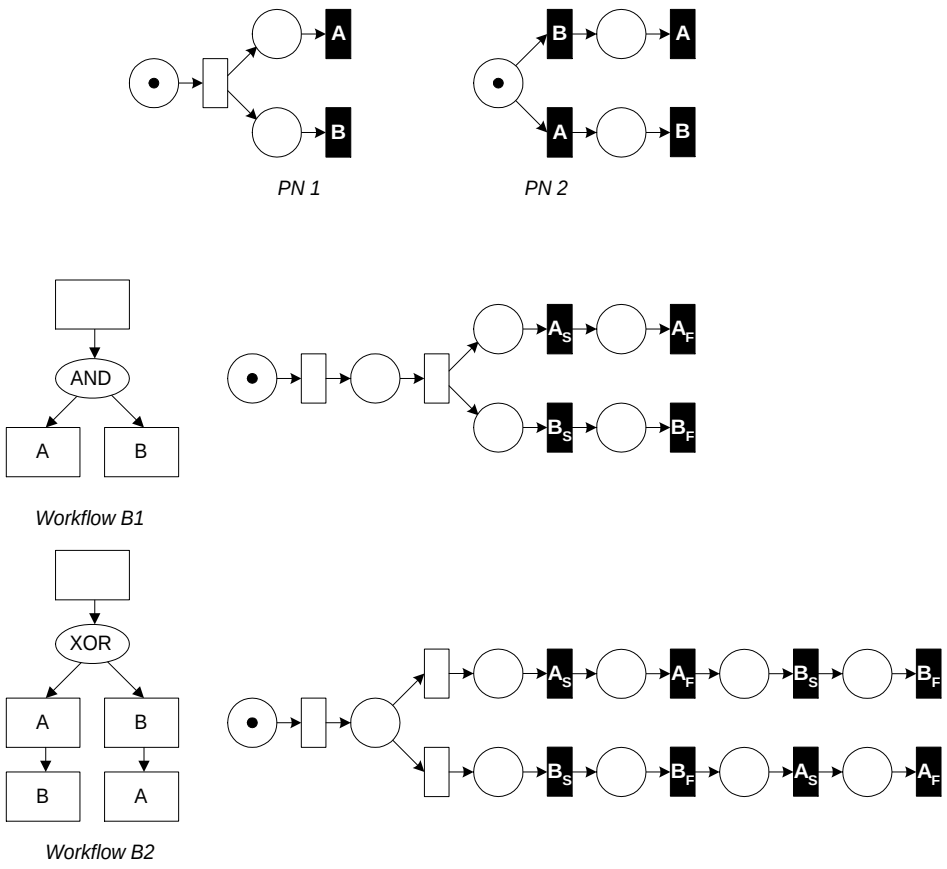

Figure 12: Interleaving vs. concurrent activity invocation

require that the *begin-end* refinements of these Petri nets be equivalent where the begin-end refinement is a refinement that replaces every labelled transition with two labelled transitions and a place that is an output to the first transition and an input to the second transition (as in Figure 12).

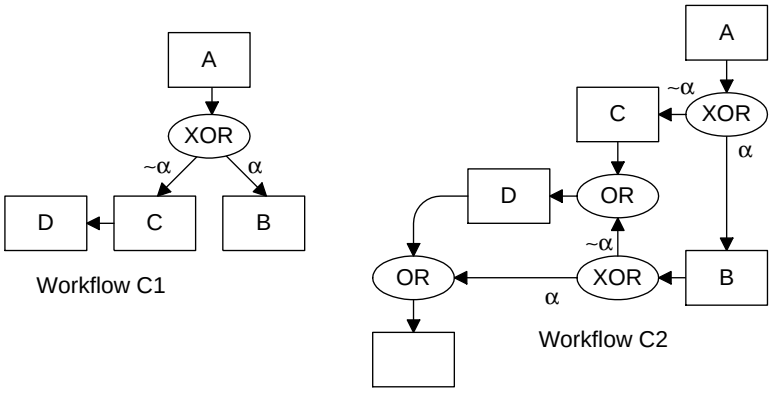

Figure 13: Equivalence in the context of data flow

Next consider Workflows $C1$ and $C2$ of Figure 13. Careful analysis of control flow taking into consideration the conditions of each of the XOR-Splits may lead to the conclusion that these workflows are equivalent. However, the Petri net mapping does not take conditions of XOR-Splits into account and Petri net representations of these two workflows are not equivalent (as in Workflow $C2$, after executing activity B it is still possible to fire transition D, which is not a possibility in Workflow $C1$). At this point the careful reader may notice that in real-world workflows activity B can change the value of α and, indeed, in Workflow $C2$ it may be possible to invoke activity D after activity B. As our paper focuses exclusively on control flow and we do not take data into consideration, we are assuming that these two workflows are not equivalent as we have no knowledge about the possible interdependencies between the two XOR-Splits in Workflow $C2$. In other words we are always treating XOR-Splits as *non-deterministic* constructs, i.e. any decision can always be taken at any point in time.

So far we have explored different notions of equivalence in a very informal manner. Our goal was to choose an equivalence notion that is relatively simple yet powerful enough to be able to distinguish workflows that need to be considered "different". To be able to establish theoretical expressiveness boundaries of different workflow classes, we need to define our equivalence notion in a formal, precise manner.

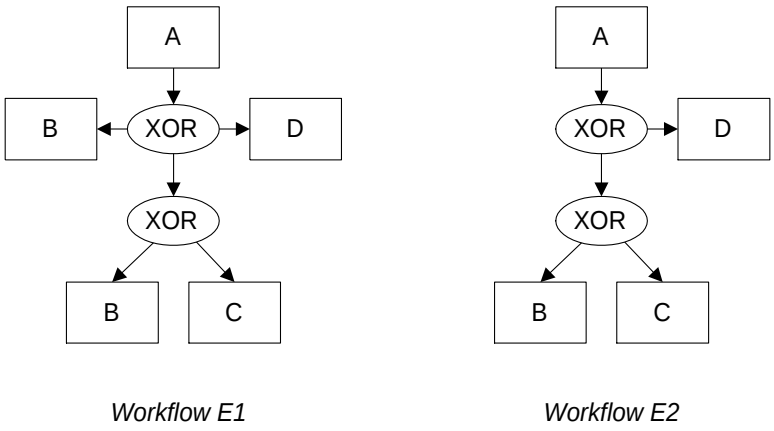

Figure 14: Weak bisimulation vs. branching bisimulation

The standard equivalence notion that is based on the interleaving assumption and preserves branching time is that of *bisimulation*. Bisimulation is extensively studied, primarily in the context of process graphs but also in the context of Petri nets. As the Petri nets that correspond to workflow models contain many silent transitions, focus is on *weak* bisimulation, where one abstracts from silent steps, i.e., silent steps may be executed but their execution is not visible for an external observer. As pointed out by van Glabbeek in [Gla94], Milner's notion of *weak bisimulation* in [Mil89] does not actually preserve branching time for silent transitions. This observation led to his introduction of the notion of *branching bisimulation*. Consider for example the two workflows of Figure 14. They are equivalent under Milner's weak bisimulation notion however they are different under van Glabbeek's branching bisimulation notion due to the fact that in workflow $E1$ there is a point where the observable run of ab diverges from the runs of ac and ad which is not the case in workflow $E2$. From a workflow point of view we would like to consider these two workflows to be equivalent due to the fact that there is no

additional data available for the second XOR-Split, therefore the moment of choice for activity B is irrelevant.

Finally, it is important that the equivalence notion distinguishes processes that successfully terminate from the ones that deadlock.

Before we introduce bisimulation formally, we would like to present a weaker equivalence notion, namely *simulation*. Understanding simulation equivalence helps with understanding bisimulation equivalence and sometimes proving simulation equivalence precedes proving bisimulation. Processes that are bisimulation equivalent are also simulation equivalent, but the reverse does not always hold.

To define (bi)simulation we adopt some of the standard notations [Mil80, Mil89, Mil99].

Definition 3.1

Let $PN = (P, T, F, L)$ be a (labelled) Petri net where L is a mapping that associates to each transition $t \in T$ a label $L(t)$ taken from some given set of actions $\mathcal{N}$. For any $a \in \mathcal{N}, M \stackrel{a}{\Longrightarrow}_{PN} M'$ means that $M \stackrel{\sigma}{\longrightarrow}_{PN} M'$ for some sequence σ of transitions, one of them being labelled with a, the others with λ; in case $a = \lambda$, the sequence can be empty. □

Definition 3.2

Let $PN = (P, T, F, L)$ be a Petri net. $M^{\varnothing}_{PN}$ is the empty marking of PN ($\forall_{p \in P} M(p) = 0$). □

Definition 3.3 (*simulation*)

Given two labelled Petri nets $PN_1 = (P_1, T_1, F_1, L_1)$ and $PN_2 = (P_2, T_2, F_2, L_2)$, a binary relation $R \subseteq \mathbb{N}^{P_1} \times \mathbb{N}^{P_2}$ is a simulation *iff*

1. *For all $(M_1, M_2) \in R$ and for each $a \in \mathcal{N}$ and M_1' such that $M_1 \stackrel{a}{\Longrightarrow}_{PN_1} M_1'$ there is M_2' such that $M_2 \stackrel{a}{\Longrightarrow}_{PN_2} M_2'$ and $(M_1', M_2') \in R$*
2. *$(M_1, M_2) \in R \Rightarrow (M_1 \stackrel{\lambda}{\Longrightarrow}_{PN_1} M^{\varnothing}_{PN_1} \Leftrightarrow M_2 \stackrel{\lambda}{\Longrightarrow}_{PN_2} M^{\varnothing}_{PN_2})$*

Net system (PN_1, M_0) can be simulated by net system (PN_2, M_0') if there is a simulation relation R relating their initial markings.

Two labelled net systems (PN_1, M_0) and (PN_2, M_0') are simulation equivalent *if (PN_1, M_0) can be simulated by (PN_2, M_0') and (PN_2, M_0') can be simulated by (PN_1, M_0).* □

Definition 3.4 (*weak bisimulation*)

Given two labelled Petri nets $PN_1 = (P_1, T_1, F_1, L_1)$ and $PN_2 = (P_2, T_2, F_2, L_2)$, a binary relation $R \subseteq \mathbb{N}^{P_1} \times \mathbb{N}^{P_2}$ is a bisimulation *iff*

1. *For all $(M_1, M_2) \in R$:*
   (a) *For each $a \in \mathcal{N}$ and M_1' such that $M_1 \stackrel{a}{\Longrightarrow}_{PN_1} M_1'$ there is M_2' such that $M_2 \stackrel{a}{\Longrightarrow}_{PN_2} M_2'$ and $(M_1', M_2') \in R$, and conversely*

(b) *For each* $a \in \mathcal{N}$ *and* M_2' *such that* $M_2 \stackrel{a}{\Longrightarrow}_{PN_2} M_2'$ *there is* M_1' *such that* $M_1 \stackrel{a}{\Longrightarrow}_{PN_1} M_1'$ *and* $(M_1', M_2') \in R$.

2. $(M_1, M_2) \in R \Rightarrow (M_1 \stackrel{\lambda}{\Longrightarrow}_{PN_1} M_{PN_1}^{\varnothing} \Leftrightarrow M_2 \stackrel{\lambda}{\Longrightarrow}_{PN_2} M_{PN_2}^{\varnothing})$

Two labelled net systems are bisimilar *if there is a (weak) bisimulation relating their initial markings.* □

Definition 3.5 (*begin-end transformation*)
Given a labelled Petri net $PN = (P, T, F, L)$ *and* $T^l = \{t \in T \mid L(t) \neq \lambda\}$, *the net* $PN^* = (P', T', F', L')$ *with*

$$\begin{array}{lcl} P' & = & P \cup \{p_t \mid t \in T^l\}, \\ T' & = & T \cup \{s_t | t \in T^l\} \cup \{f_t | t \in T^l\} \setminus T^l \\ F' & = & (F \cap (P' \times T' \cup T' \times P')) \cup \\ & & \{(p, s_t) \mid p \in {\bullet t} \wedge t \in T^l\} \cup \{(s_t, p_t) \mid t \in T^l\} \cup \\ & & \{(p_t, f_t) \mid t \in T^l\} \cup \{(f_t, q) \mid q \in t{\bullet} \wedge t \in T^l\} \\ L' & = & \{(t, \lambda) \mid t \notin T^l\} \cup \{(s_t, L(t)_{\mathsf{S}}) \mid t \in T^l\} \cup \{(f_t, L(t)_{\mathsf{F}}) \mid t \in T^l\} \end{array}$$

is the begin-end transformation of PN. □

Definition 3.6 (*workflow equivalence*)
Workflow models $\mathcal{W}_1$ *and* $\mathcal{W}_2$ *are* equivalent *iff the begin-end transformations of their corresponding net systems are bisimilar.* □

Sometimes we will compare workflow models with net systems. In that case we will say that a workflow model $\mathcal{W}$ is equivalent to a net system PN iff the begin-end transformations of the corresponding net system of $\mathcal{W}$ and PN are bisimilar.

4 Basic Expressiveness Results

This section will establish precise characterizations of the expressive power of Standard Workflow Models (Section 4.1), Safe Workflow Models (Section 4.2), and Synchronizing Workflow Models (Section 4.3). It is important to differentiate between *suitability* and *expressive power*. Our work on workflow patterns [ABHK00, AHKB00, AHKB02, WPH02] focuses on suitability issues, e.g., Does a workflow language offer *direct support* for a pattern frequently appearing in workflow designs? In this section we focus on the expressive power. The question is not whether there is direct support for a pattern but whether it is possible to express certain constructs in some way (i.e., direct or indirect) under the notion of equivalence introduced in the previous section. Answering such a question is far from trivial. Consider for example the work on Task Structures [HOR98, HO99] which ignores the fact that Task Structures without decomposition are less expressive than Petri nets. (Hence its results need to be reconsidered.) Therefore, we carefully formulate and prove some basic expressiveness results.

4.1 Standard Workflow Models

This subsection focuses on the expressive power of Standard Workflow Models. It is easy to verify that the corresponding Petri net system of a Standard Workflow Model is free-choice and one may wonder if these models have the same expressive power as free-choice Petri nets. This turns out not to be true. Standard Workflow Models are in fact less expressive than free-choice Petri nets. This result is not merely of theoretical importance. We will show that the Deferred Choice pattern introduced in [ABHK00, AHKB00, AHKB02] cannot be modelled using Standard Workflow Models.

Theorem 4.1 Standard Workflow Models are less expressive than free-choice Petri nets.

Proof:

First observe that any corresponding net of a Standard Workflow Model is free-choice. To complete the proof, we have to find a free-choice Petri net that does not have an equivalent Standard Workflow net. Such a net is shown in Figure 15. As can be seen, this free-choice Petri net, which we will refer to as PN_d, is very simple, yet its inherent properties may be overlooked in workflow analysis.

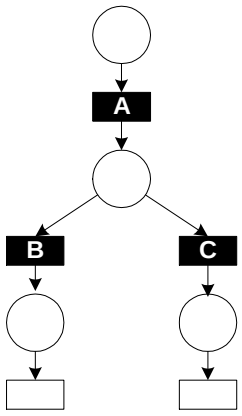

Figure 15: Free-choice Petri net with deferred choice

Suppose there exists a Standard Workflow net, say PN_s, equivalent to PN_d. Let us focus on marking M_1^d where there is a token in the place input to the transitions labelled B and C. If PN_d is to be bisimulation equivalent to PN_s, there should be a marking M_1^s that is related through the bisimulation relation to M_1^d (see Figure 16). The first observation is that in M_1^s it is not possible that both B and C are enabled. The reason for this is that although markings in Standard Workflow Models can exist which enable more than one labelled transition, it is not possible that the firing of one labelled transition leads to other labelled transitions being disabled (cf. Standard Workflow Models do not allow for the construct shown on the right in Figure 2 and model the XOR-Split as shown in the middle). Hence, if both transitions B and C are enabled in M_1^s, they will both be executed at some stage, and as this is not the case for M_1^d, these two markings cannot be related through a bisimulation relation.

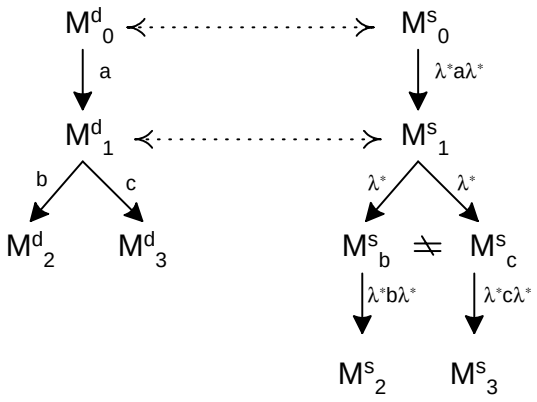

Figure 16: Illustration of bisimulation relations between markings

For M_1^s then to be related to M_1^d through the bisimulation relation, it should be possible to reach markings that enable B and markings that enable C. As transitions labelled B and C cannot be enabled at the same time, we have that at least one silent step is needed (from M_1^s) to reach either a marking in which a transition labelled B is enabled or a marking in which a transition labelled C is enabled. Without loosing generality, we can assume that at least one silent step is needed to reach a marking in which a transition labelled B is enabled. Let us refer to such a marking as M_b^s. Through the bisimulation relation, this particular marking has to be related to marking M_1^d in PN_d. However, in M_1^d the transition labelled C is enabled, while in M_b^s C cannot be performed anymore. Contradiction. □

Naturally, the previous result immediately raises the question as to what the exact expressive power of Standard Workflow Models is. Before we provide a complete characteristic of the expressive power of Standard Workflow Models let us focus on some of the most basic properties of theses models.

The following lemma states that once a process element becomes enabled, it cannot be disabled by firing any other process element but itself and can be proved by case distinction. (Note that this lemma was already used implicitly in Theorem 4.1.)

Lemma 4.1 Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Standard Workflow Model and $e, p \in \mathcal{P}$ enabled process elements of $\mathcal{W}$ in a given instance of $\mathcal{W}$ ($e \neq p$). After firing p, e is still enabled.

From all the process elements only activities contain labelled transitions. The next theorem proves that for a free-choice Petri net to have a bisimulation equivalent Standard Workflow Net it is sufficient that all its labelled transitions, once they become enabled, cannot be disabled by firing any other transitions but themselves.

We will refer to such a subclass of free-choice Petri nets (see Appendix A for a formal definition of a free-choice Petri net) as Free-Choice Deterministic Action Nets.

Definition 4.1

A Free-Choice Deterministic Action Net *(FCDA net)* $PN = (P, T, F, L)$ *is a labelled free-choice Petri net where every labelled transition has exactly one input place and that place is not an input to any other transition:* $\forall_{t \in T} [L(t) \neq \lambda \Rightarrow \forall_{t' \in T} [\bullet t \cap \bullet t' \neq \varnothing \Rightarrow t = t']]$.

□

Theorem 4.2 Standard Workflow nets are as expressive as FCDA net systems.

Proof:

As every Standard Workflow net is an FCDA net, we can focus on proving that every FCDA net has a bisimilar Standard Workflow net. This will be achieved in a constructive way, i.e. the proof will focus on the translation of any arbitrary FCDA net to a Standard Workflow net. The organization of the proof is as follows: given an FCDA net, PN we will perform a number of bisimulation-preserving transformations on it eventually deriving a net, PN_1. At the same time we will construct a Standard Workflow Model $\mathcal{W}$ for which its corresponding Petri net $PN_{\mathcal{W}}$ is identical to PN_1. This will conclude the proof.

The translation takes a number of steps. In intermediate stages, instead of a pure Petri net notation we will use a shorthand representation of Petri net subnets using workflow construct notation. This serves two purposes: (1) it simplifies the complexity of the derived net and (2) it allows us to construct the desired Standard Workflow Model. An example of a shorthand notation is shown in Figure 17, which shows three places linked to a hybrid Activity and AND-Join construct. AND-Split, XOR-Split and OR-Join constructs are derived in a similar manner. All presented translations will make sure that hybrid constructs will always be linked to places or to each other. Let us define two sets, T^* and P^* representing the sets of transitions and places respectively that are part of the hybrid net but not part of a hybrid structure. Initially $T^* := T$ and $P^* := P$. Each transformation step aims to reduce the number of elements in T^* or P^* (or both) until all transitions and places are part of a hybrid structure. For example, in Figure 17 $T^* = \varnothing$ while $P^* = \{P1, P2, P3\}$.

For the construction to be meaningful, it is required that every transformation step preserves equivalence. This is straightforward to check for each of the steps presented.

The following steps describe the procedure to transform any arbitrary FCDA system into a bisimulation equivalent Standard Workflow net.

1. Replace all places with initial tokens with the structure shown in Figure 18. The number of null activities should correspond to the number of tokens. If there is only one token then the OR-Join is redundant and can be omitted. After this step there are no tokens in any of the places of the net. This step does not affect T^* or P^*.
2. A labelled transition has exactly one input place that is not shared with any other transition. Diagram (a) of Figure 19 presents the transformation for a labelled transition with one output place and diagram (b) of that figure presents the transformation for a labelled transition with many output places. After this step, there are no labelled transitions anymore, i.e. $T^* = \{t \in T \mid L(t) = \lambda\}$.

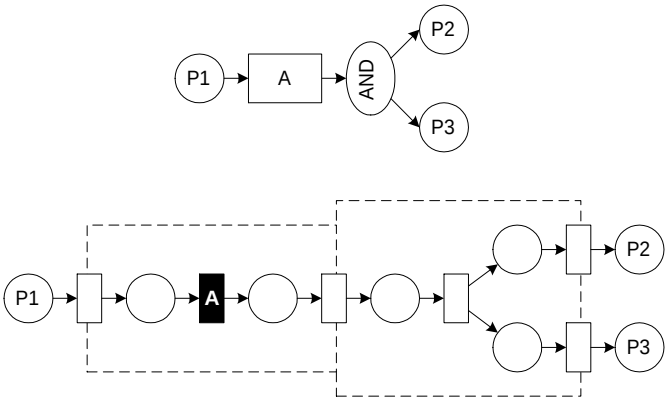

Figure 17: Interpretation of a sample hybrid net

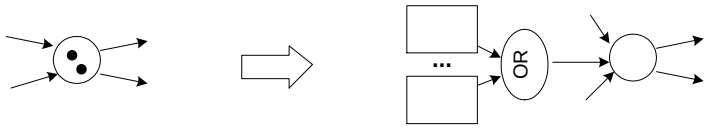

Figure 18: Translation of marked places

3. Replace transitions with no input or output places and places with no input or output transitions by corresponding structures as shown in Figure 20. Note that the semantics of Splits without incoming transitions and Joins without outgoing transitions is such that these transformations are equivalence preserving. After that step

$$T^* = \{t \in T \mid L(t) = \lambda \wedge |t\bullet| \geq 1 \wedge |\bullet t| \geq 1\}$$
$$P^* = \{p \in P \mid |p\bullet| \geq 1 \wedge |\bullet p| \geq 1\}$$

4. Replace transitions that have the same, nonsingular, set of input places with the structure shown in Figure 21. Effectively, from this step onwards, if transitions share any input places, they share exactly one (remember that an FCDA net is free-choice). Note that if any of the transitions have only one output place, the

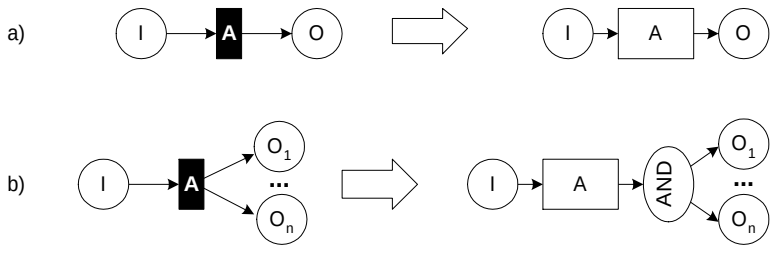

Figure 19: Translations of labelled transitions

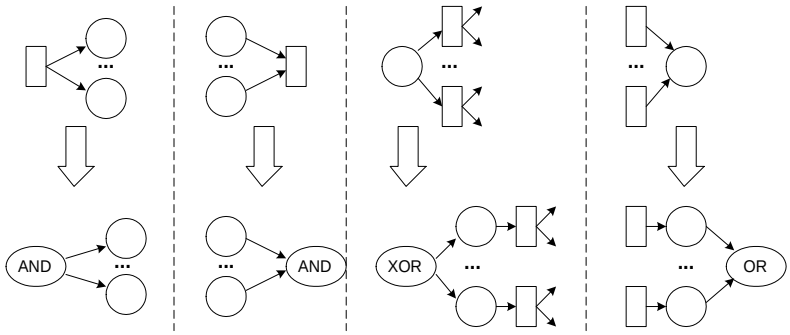

Figure 20: Translations of transitions/places without input or output

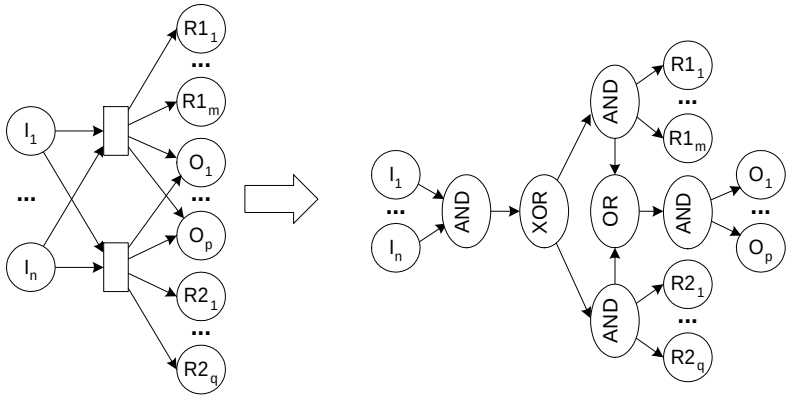

Figure 21: Removal of transitions sharing nonsingular set of input places

AND-Split can be omitted. Formally we now have that

$$\begin{aligned} T^* &= \{t \in T \mid L(t) = \lambda \wedge |t\bullet| \geq 1 \wedge |\bullet t| \geq 1 \wedge \\ &\qquad \forall_{t' \in T}\,[\bullet t \cap \bullet t' \neq \varnothing \Rightarrow (t = t' \vee |\bullet t| = 1)]\} \\ P^* &= \{p \in P \mid |p\bullet| \geq 1 \wedge |\bullet p| \geq 1\} \end{aligned}$$

5. At this stage it is still possible that transitions share input places, output places, or both. In Figure 22 the removal of such transitions is defined in diagrams (a), (b) and (c). Again, in all these transformations, if any of the transitions have only one input or one output place, the AND-Joins and AND-Splits respectively can be omitted. After this step:

$$\begin{aligned} T^* &= \{t \in T \mid L(t) = \lambda \wedge |t\bullet| \geq 1 \wedge |\bullet t| \geq 1 \wedge \\ &\qquad \forall_{t' \in T}\,[(\bullet t \cap \bullet t' \neq \varnothing \vee t\bullet \cap t'\bullet \neq \varnothing) \Rightarrow t = t']\} \\ P^* &= \{p \in P \mid |p\bullet| \geq 1 \wedge |\bullet p| \geq 1\} \end{aligned}$$

6. In this step all remaining transitions are removed as shown in Figure 23. There are four possibilities - the transition may have one input place and many output places, many input places and one output place, many input and many output places, or

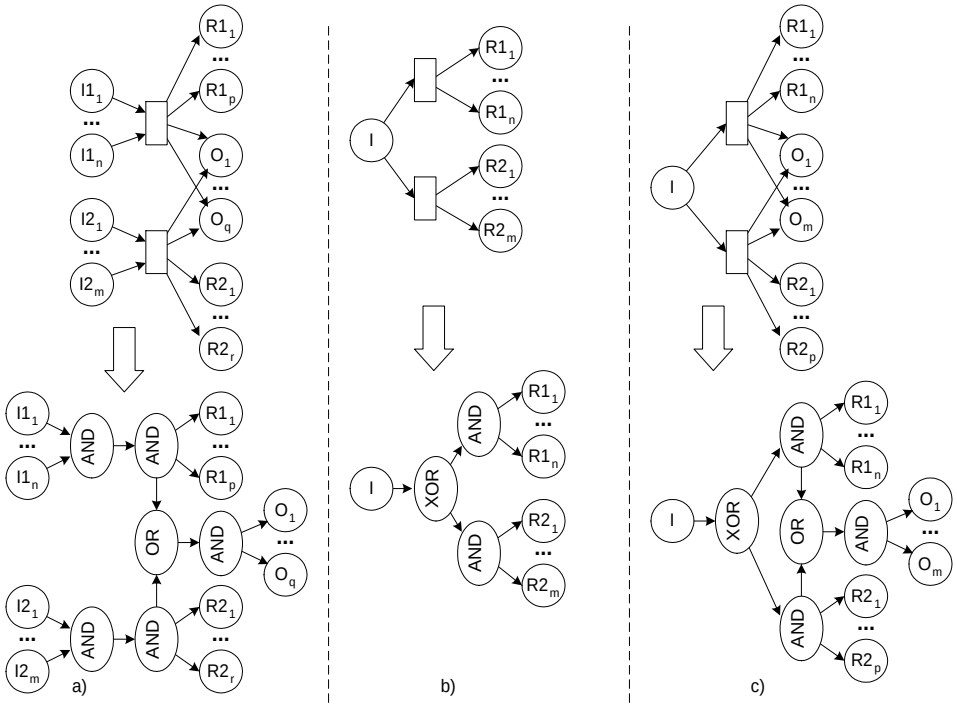

Figure 22: Removal of transitions sharing input or output places

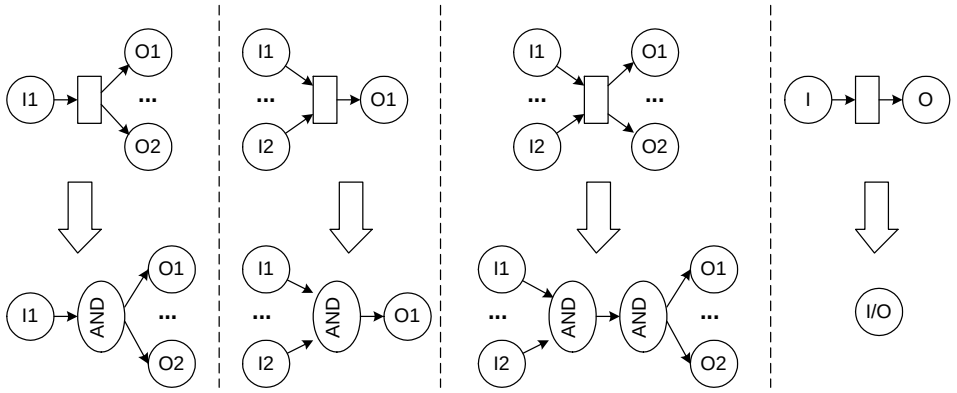

Figure 23: Removal of transitions

one input place and one output place. After this step

$$\begin{aligned} T^* &= \varnothing \\ P^* &= \{p \in P \mid |p\bullet| \geq 1 \wedge |\bullet p| \geq 1\} \end{aligned}$$

7. Now that all transitions are removed, places can only be linked to workflow constructs. They can subsequently be removed according to the schema shown in Figure 24. Again, as with the previous step, there are only four possibilities. After this step the net consists entirely of hybrid constructs, i.e. $T^* = \varnothing$ and $P^* = \varnothing$.

As every step that we have taken so far is equivalence preserving, the hybrid net that we

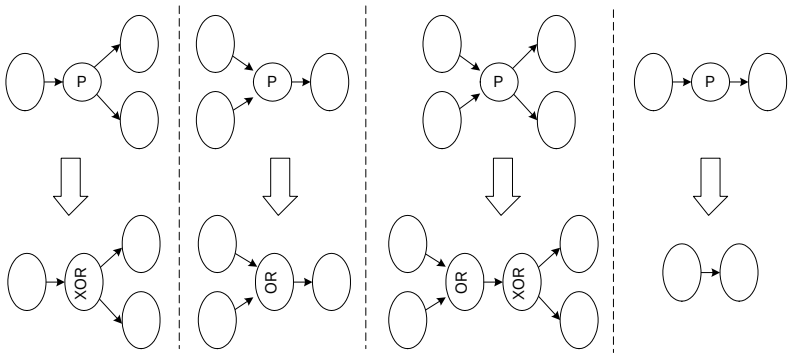

Figure 24: Removal of places

have constructed, PN_H, is equivalent to our source FCDA net, PN. As PN_H consists entirely of hybrid structures, it is possible to construct a Standard Workflow Model $\mathcal{W}$ by replacing hybrid structures with the corresponding workflow constructs. The corresponding Petri net PN_W of the Standard Workflow Model $\mathcal{W}$ constructed in such a manner is identical to PN_H. As PN_H is equivalent to PN, it follows that W is equivalent to PN which concludes the proof.

□

Example 4.1 An example of the transformation described in the proof of Theorem 4.2 of an FCDA net to a Standard Workflow Model is shown in Figure 25. Obviously, the Standard Workflow Model can be further reduced (the final AND-Join is redundant and can be removed), however, this is of no importance in this context. Note that the FCDA net and the Petri net corresponding to the Standard Workflow Model (see Figure 5) are indeed weak bisimulation equivalent. □

4.2 Safe Workflow Models

As explained in Section 2, the main difference between Standard Workflow Models and Safe Workflow Models is in the interpretation of the OR-Join in case it is triggered by more than one incoming branch (as could e.g. happen in case an OR-Join follows an AND-Split).

In this section we would like to answer the question whether this evaluation strategy limits the expressive power of the workflow language. Formally, this translates to the question as to whether it is possible to transform any given Standard Workflow Model to an equivalent Safe Workflow Model. A technique typically required for this is node replication (illustrated in Figure 26).

Node replication can be compared to net unfolding as described in for example [GV87]. The unfolded net can be thought of as the safe version of the original net. Unfolding as described in [GV87] preserves bisimulation equivalence.

It is immediately clear that if the original net is not bounded, then the unfolding is infinite. Hence, it is impossible to convert a Standard Workflow Model that may result in an unlimited

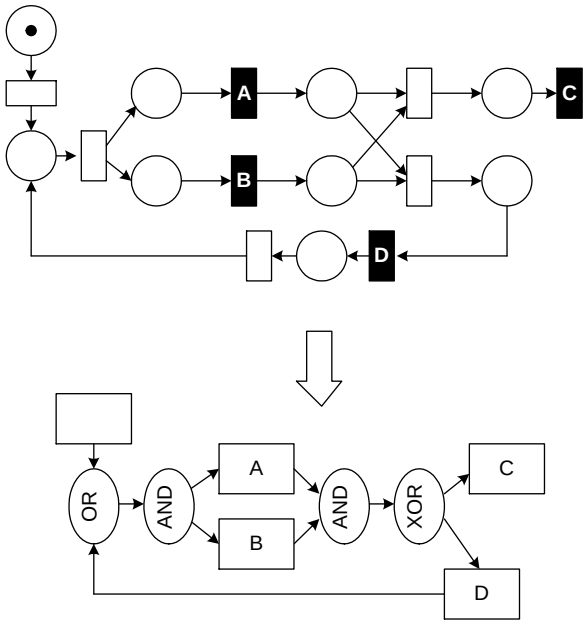

Figure 25: FCDA net with the corresponding Standard Workflow Model

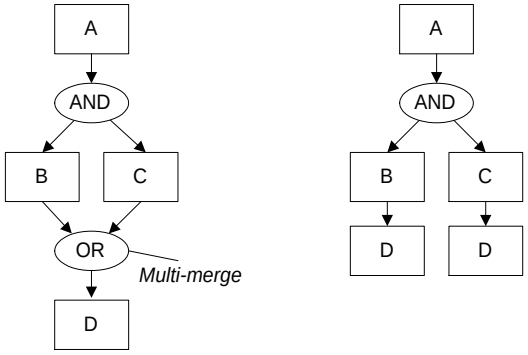

Figure 26: Node replication

number of multiple instances of some activity, into a finite Safe Workflow Model. Therefore, let us focus on bounded workflow models.

It is always possible to convert a bounded Petri net into an equivalent safe Petri net by unfolding.[5] However, from a workflow perspective, the fundamental problem with this technique is that unfolding as presented in [GV87] may transform a free-choice Petri net into a net which is not free-choice. The next theorem demonstrates that this is a true problem which cannot be circumvented. There exist bounded Standard Workflow specifications that do not have a safe equivalent.

Before presenting a proof we would like to introduce two lemmas. The first one captures one of

[5]Replace a k-bounded place by $k+1$ safe places indicating the number of tokens in the original place. Each input/output transition is replicated k times to account for the different circumstances in which a token is produced/consumed.

the important characteristics of free-choice nets. This lemma will be used in several subsequent proofs and it states that if there is a path from a place q to a place p, and $[p]$ is a home marking[6], then if q contains a token it can be moved to p by a firing sequence containing all transitions on the path between q and p.

Lemma 4.2 Let $PN = (P, T, F, M_0)$ be a live and bounded free-choice Petri net with a home marking $M_0 = [p]$ (i.e. the state marking a place p). Let M be a reachable marking which marks place q and let $x =< p_1, t_1, p_2, t_2, ..., t_{n-1}, p_n >$ with $p_1 = q$ and $p_n = p$ be an acyclic directed path in the net. Then there is a firing sequence σ such that $M \xrightarrow{\sigma} [p]$, each of the transitions $\{t_1, ..., t_{n-1}\}$ is executed in the given order, and none of the intermediate markings marks p.

Proof:

If $p = q$ then the lemma holds. If $p \neq q$ then there is a firing sequence removing the token from q (since $[p]$ is a home marking). Let $\sigma_1 t$ be the firing sequence removing the token from q, i.e. $t \in q\bullet$. Let M_1 be the marking enabling t, i.e. $M \xrightarrow{\sigma_1} M_1$. As the net is free-choice and t is enabled in M_1, t_1 is also enabled in M_1 (recall that $q \in \bullet t_1$ and $q \in \bullet t$ implies $\bullet t_1 = \bullet t$). It is therefore possible to fire t_1, i.e. $M_1 \xrightarrow{t_1} M_2$. In M_2 place p_2 is marked (as $p_2 \in t_1\bullet$).

By recursively applying the argument to the remaining places and transitions it is possible to construct a firing sequence σ such that each transition in $\{t_1, ..., t_{n-1}\}$ occurs and $M \xrightarrow{\sigma} [p]$, i.e. it is possible to execute the transitions in the order of the directed path between q and p.

Finally, we need to prove that none of the intermediate markings reached by executing σ marks p. Suppose that p was marked before completing σ. There is a token moving from q to p via path $< q, t_1, p_2, t_2, ..., t_{n-1}, p >$. Therefore, for any intermediate marking there is a token in one of the places $\{p_2, ..., p_{n-1}\}$. However, if p and some other place are marked at the same time the net is unbounded. (This follows from the well-known Boundedness lemma, cf. Lemma 2.22 [DE95].) This contradiction completes the proof.

□

Figure 27 illustrates two Petri nets, PN_1 being a free-choice net, and PN_2 not. Place p is a home marking for both nets. Let us concentrate on the path from place q to place p containing transitions t_1 and t_2. In net PN_1, from any marking having a token in place q it is possible to fire transitions t_1 and t_2. In net PN_2 that is not always the case as the marking shown illustrates.

The second lemma introduces a construct that we would like to refer to as a "selective synchronizer". Such a synchronizer has three incoming transitions. In the context shown in Figure 28 the Selective Synchronizer awaits completion of activity A and either activity B or activity C. Depending on whether B or C completes, activity D or activity E respectively is enabled. It is worth noticing that the desired behaviour is not achievable using standard workflow constructs. For example, had we put an OR-Join after activities B and C, it would not be possible

[6] $[p]$ is the marking with just one token in place p and a home marking is a marking which is reachable from every marking reachable from the initial state (cf. Appendix A).

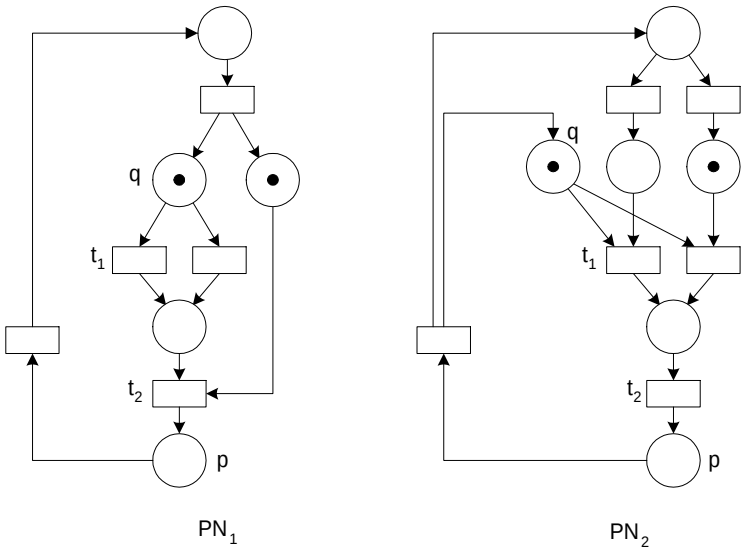

Figure 27: Illustration of Lemma 4.2

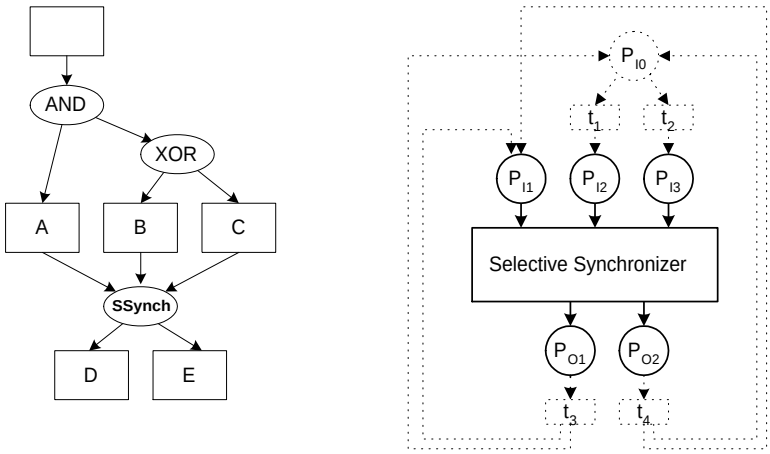

Figure 28: Illustration of Lemma 4.3

to make a correct choice between D and E. On the other hand any attempts to use a standard AND-Join construct leads to a deadlock.

The following lemma defines the "selective synchronizer" in a formal way and proves that this construct is inherently non free-choice.

Lemma 4.3 Let $PN = (P, T, F)$ be the Petri net as shown (in bold lines) in the right diagram of Figure 28. The Selective Synchronizer construct cannot be free-choice if:

- Any marking with tokens in any of the places p_{O1}, or p_{O2} has one token in exactly one of these places and no other places. Such markings are called *output markings*;
- – From $p_{I1} + p_{I2}$, the only reachable output marking is p_{O1};

– From $p_{I1} + p_{I3}$, the only reachable output marking is p_{O2}.

Proof:

Consider the Selective Synchronizer net augmented with place p_{I0}, transitions t_1, t_2, t_3 and t_4 and arrows as shown with dashed lines in Figure 28. The resulting Petri net is called the short-circuited net. Clearly, $[p_{I1}, p_{I2}]$, $[p_{I1}, p_{I3}]$, $[p_{O1}]$ and $[p_{O2}]$ are home markings. We can assume that the Selective Synchronizer construct contains no dead transitions and that the short-circuited net is strongly connected. Places and transitions without any input and/or output arcs are either inactive and do not contribute to the external behaviour or are conflicting with the requirements. As a result, the short-circuited net is live and bounded with home markings $[p_{I1}, p_{I2}]$, $[p_{I1}, p_{I3}]$, $[p_{O1}]$ and $[p_{O2}]$.

As the marking $p_{I1} + p_{I2}$ is followed by $[p_{O1}]$, we can conclude that there must be a path from p_{I1} to p_{O1} and from p_{I2} to p_{O1}. Similarly there must be a path from p_{I1} to p_{O2} as the marking $p_{I1} + p_{I3}$ is followed by $[p_{O2}]$.

Suppose that the selective synchronizer is a free-choice construct. The short-circuited net is then free-choice too (the only choice which is added or changed is the choice involving p_{O1}). According to Lemma 4.2 if there is a path from p_{I1} to p_{O2}, then there is also a firing sequence leading from $p_{I1} + p_{I2}$ to p_{O2} not marking p_{O2} in-between. Note that this firing sequence does not involve any of the newly added transitions t_1, t_2, t_3 and t_4. Therefore, the firing sequence is also possible in the original net. This is contradictory with the assumptions. Therefore, the synchronizer cannot be free-choice. □

Finally we are ready to present a theorem that shows the expressiveness limitation of the safe evaluation strategy.

Theorem 4.3 (*limited power of the safe evaluation strategy*) There exist bounded Standard Workflow Models without a deadlock, for which there exists no equivalent Safe Workflow Model.

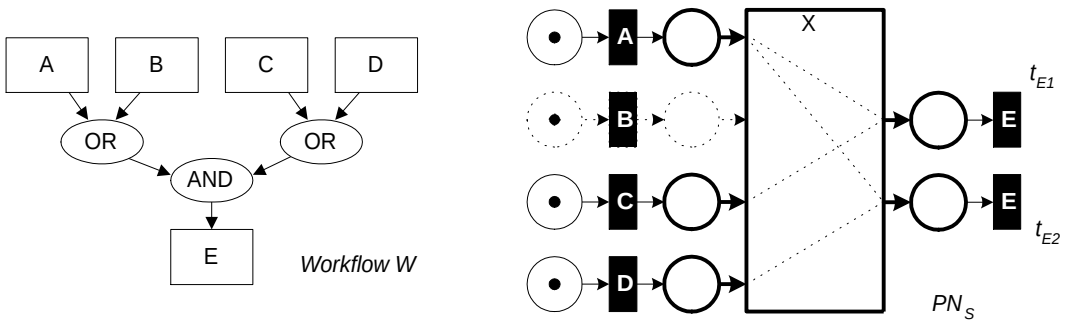

Figure 29: Multiple instances specification

Proof:

Consider the deadlock free and bounded Standard Workflow Model $\mathcal{W}$ in Figure 29.

There are four initial activities named A, B, C, and D. The activity named E can be fired after either A and C have been completed, or A and D, or B and C or B and D. Subsequently activity E can be fired for the second time when the remaining activities are completed.

Let $\mathcal{S}$ be a Standard Workflow Model that is bisimulation equivalent to $\mathcal{W}$ and $PN_{\mathcal{S}}$ be the corresponding net of $\mathcal{S}$. For $\mathcal{S}$ to be bisimulation equivalent to $\mathcal{W}$, $PN_{\mathcal{S}}$ needs to have transitions labelled A, B, C, and D as well as at least two transitions labelled E. The last requirement comes from the fact that in workflow $\mathcal{W}$ it is possible to enable and fire the transition labelled E twice in a concurrent manner.

In $\mathcal{W}$ it is possible to enable activities A, B, C and D concurrently. Hence there must be a reachable marking M of $PN_{\mathcal{S}}$ that enables transitions labelled A, B, C and D and no other labelled transitions. Let us call these transitions t_A, t_B, t_C and t_D respectively.

In $\mathcal{W}$ it is possible to fire activities A and C followed by activity E. Thus in net PN_S there must be a path from t_A and t_C to a transition labelled E. Let us call this transition t_{E1}.

Similarly there must be paths from transitions t_A and t_D to a transition labelled E as well as paths from t_B, t_C and t_B, t_D to transitions labelled E. Let us call these transitions t_{E2}, t_{E3} and t_{E4} respectively.

Consider transitions t_{E1}, t_{E2} and t_{E4}. Transitions t_{E1} and t_{E4} cannot be the same (otherwise the net would not be safe) whereas it is possible that $t_{E1} = t_{E2}$ or $t_{E4} = t_{E2}$. Without loss of generality, suppose that t_{E2} is such that $t_{E1} \neq t_{E2}$.

Any labelled transition in $PN_{\mathcal{S}}$ needs to have exactly one input and one output place. Output places of transitions t_A, t_C, t_D and input places of transitions t_{E1} and t_{E2} along with the subnet X shown in the right diagram of Figure 29 form a subnet that fulfils the requirements of Lemma 4.3 (Selective Synchronizer), hence the subnet X and subsequently net PN_S cannot be free-choice. This contradicts the assumption that PN_S is the corresponding net of a Standard Workflow Model. □

Theorem 4.3 shows that the choice for a safe execution strategy limits the expressive power of the corresponding workflow engine, even if one is only interested in bounded deadlock-free workflows. A practical example of a process that might need the type of synchronisation shown in Figure 29 is a process in which activities A and B represent the manufacturing of an item of type X, activities C and D the manufacturing of an item of type Y and activity E represents the assembling of an item of type X and an item of type Y.

4.3 Synchronizing Workflow Models

This section concentrates on a precise characterization of the expressive power of Synchronizing Workflow Models. To this end, we start with discussing some elementary properties.

First it is important to observe that arbitrary loops (called Arbitrary cycles in [ABHK00, AHKB00, AHKB02, WPH02]) would cause problems in Synchronizing Workflow Models. Consider for example an activity A which is to trigger an activity B, while there is a trigger back from B to A, i.e., there is a causal dependency from A to B and from B to A. Activity A

can only be executed if all its incoming triggers have been evaluated. However, one of these triggers depends on activity B, which on its turn depends on activity A resulting in immediate deadlock. For this reason only acyclic Synchronizing Workflow Models are considered in the remainder of this section.

Synchronizing Workflow Models have the property that every process element will receive exactly one token, true or false, for each of its input branches, and as a result it will produce a token for each of its outgoing branches. In Petri net terms this means that for every process element e of the model, exactly one of the corresponding transitions AT_e or AF_e (for an activity) will fire once. This result then effectively shows that Synchronizing Workflow Models are safe and never deadlock. Before the proof is presented let us first present some fundamental properties of Synchronizing Workflow Nets.

The following lemma can be proved by case distinction.

Lemma 4.4 Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Synchronizing Workflow Model and $e \in \mathcal{P}$ be a process element of $\mathcal{W}$ that is enabled in a reachable marking M (cf. Definition 2.23) of the corresponding net system of $\mathcal{W}$, then:

1. There is a transition of the associated net of e which is enabled in M;
2. Firing this transition results in a marking where e is completed.

While the above lemma provides a sufficient condition for at least one of the transitions associated with a process element to be enabled, the following lemma shows that this condition is also necessary (again the proof can be given using case distinction).

Lemma 4.5 Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Synchronizing Workflow Model, $e \in \mathcal{P}$ a non-initial process element of $\mathcal{W}$ and $x \in in(e)$. Then for any marking M of the corresponding net of $\mathcal{W}$ such that $M(rt_{e,x}) = 0 \wedge M(rf_{e,x}) = 0$, none of the transitions in $T^e_{\mathcal{W}}$ is enabled.

Theorem 4.4 Let $\mathcal{W} = (\mathcal{P}, \mathsf{Trans}, \mathsf{Name})$ be a Synchronizing Workflow Model. Any process element $e \in \mathcal{P}$ in this model will fire exactly once.

Proof:

By induction over the depth n of process elements, where the depth of the process element is defined as the *longest* path from this process element to a process element without incoming transitions.

The case of $n = 0$ is obvious. Indeed, any process element with no incoming branches is initially enabled, and they will fire exactly once as they cannot be enabled again after they have fired.

For the induction step consider an arbitrary process element p at depth n. All its input elements have a depth less than n, hence it can be assumed that they will fire exactly once. According to Lemma 4.5, process element p cannot fire before all input elements have actually fired. Once this has happened, process element p is enabled (Lemma 4.4). As an enabled process element cannot be disabled by firing other process elements, process element p will indeed eventually fire. It cannot be re-enabled as its input elements will never fire again. Hence it can be concluded that process element p will fire exactly once.

□

Corollary 4.1 Synchronizing Workflow Models are safe.

Corollary 4.2 Synchronizing Workflow Models do not have a deadlock.

In the remainder of this section, focus is on the expressive power of Synchronizing Workflows in relation to Standard Workflows. We will show that for any acyclic, well-behaved Standard Workflow Model there is an equivalent Synchronizing Workflow Model. We restrict ourselves to acyclic models as in our definition of Synchronizing Workflow Models cycles are not allowed and we have not made provision for the formal specification of iterative behaviour through decomposition, cf. [Kie02]. Similarly only well-behaved models are considered as according to Theorem 4.4 Synchronizing Workflow Models never result in deadlock and are always safe.

Definition 4.2
A WB-system *is a labelled Petri net system which corresponds to an acyclic, well-behaved Standard Workflow Model.* □

The following proposition captures the formal properties of WB-systems.

Proposition 4.1 A WB-system $\mathcal{P} = (P, T, F, L, M_0)$ has the following properties:

- There are no sink places (i.e. a place p such that $p\bullet = \varnothing$);
- The net is free-choice;
- Every node $x \in P \cup T$ is on a path from a source place (i.e. a place p such that $\bullet p = \varnothing$);
- The net is safe starting from the initial marking with just tokens in the source places;
- There are no dead transitions starting from the initial marking with just tokens in source places;
- From any marking reachable from the initial marking with just tokens in source places, it is possible to reach the empty marking.

The first three properties are syntactical and can be derived from the corresponding definitions. The fourth property follows from the requirement that $\mathcal{P}$ is well-behaved. The fifth property is more involved but can be derived from the fact that the net is free-choice, there is a path from a source transition for any transition, and the empty marking can always be reached. The last property follows directly from the fact that the corresponding Standard Workflow Model is terminating.

The results that follow summarise some important characteristics of WB-systems. These will be useful for providing a formal relationship between acyclic well-behaved Standard Workflow Models and Synchronizing Workflow Models.

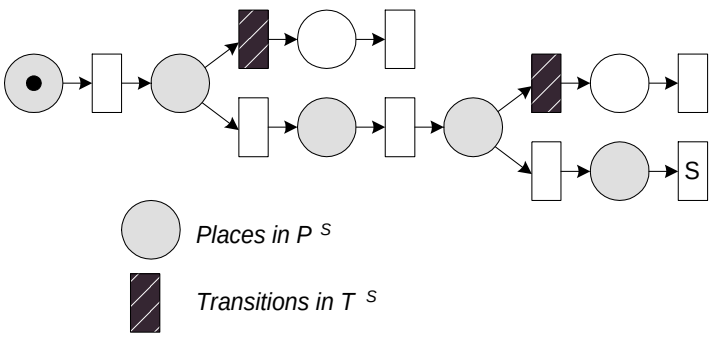

Figure 30: Example of sets P^s and T^s

Definition 4.3

Let $\mathcal{P}$ be a WB-system and $S_{\mathcal{P}}$ the set of its sink transitions, i.e. $S_{\mathcal{P}} = \{t \in T \mid t\bullet = \varnothing\}$. For any $s \in S_{\mathcal{P}}$, P^s is the set of places from which s is reachable by following the arcs in F, i.e. for each place $p \in P^s$ there is a directed path from p to s, and T^s is the set of transitions which consumes tokens from P^s but does not produce any token for P^s.

□

Example 4.2 A simple example of the above definition is depicted in Figure 30. □

Lemma 4.6 Let $\mathcal{P}$ be a WB-system. Whenever a place $p \in P^s$ is marked, s can fire, i.e. there is a firing sequence enabling s.

Proof:

Let M be a marking that marks place p. If $p \in \bullet s$ then the lemma holds since, as the net does not deadlock and is free-choice it is always possible to fire transition s. Let $x =< p_1, t_1, ..., p_n, t_n >$ be a directed path with $p_1 = p$ and $t_n = s$. As the net does not deadlock and is free-choice, there must be a firing sequence that enables transition t_1. Firing t_1 marks place p_2. By recursively applying the argument to the remaining places and transitions it is possible to construct a firing sequence σ such that $M \xrightarrow{\sigma} M'$, and M' is a marking that marks a place q such that $q \in \bullet s$. □

Lemma 4.7 Let $\mathcal{P}$ be a WB-system. Transitions in $S_{\mathcal{P}}$ can fire only once.

Proof:

If a sink transition can fire twice, it is possible to delay the first firing until the second one and clearly the WB-system is not safe in that case. □

Lemma 4.8 Let $\mathcal{P}$ be a WB-system. Firing a transition from T^s permanently disables s.

Proof:

To prove this it is shown that the places in P^s become unmarked after firing a transition

in T^s. Consider a place $p_1 \in P^s$ which contains a token which can be removed by firing a transition t in T^s and another place $p_2 \in P^s$ which remains marked after firing t. Suppose that t fires, then, based on Lemma 4.6, there is a firing sequence enabling s. If t does not fire, the same firing sequence is enabled. (Note that the tokens produced by t are not needed to enable any transition on a path to s.) However, this implies that after executing this sequence, p_1 is still marked, and based on Lemma 4.6, s could fire again. This is not possible as indicated by Lemma 4.7. Therefore, all places in P^s become unmarked after firing a transition in T^s. □

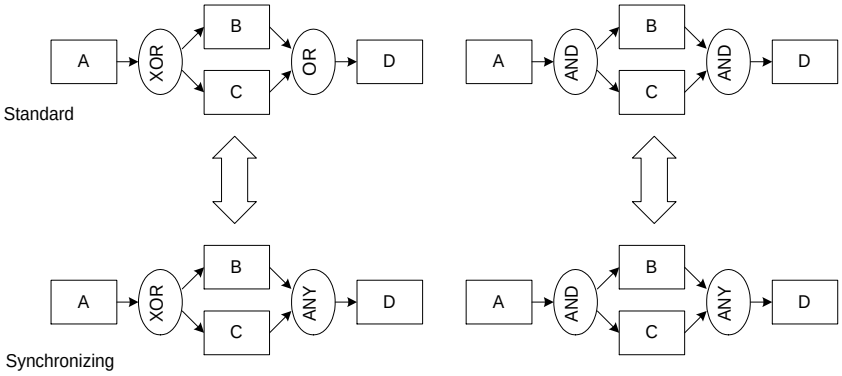

Figure 31: Equivalent Standard and Synchronizing Workflows

In order to examine the expressive power of Synchronizing Workflow Models, it is important to fully understand the expressive power of its ANY-Join construct. To this end, consider the workflow depicted in Figure 31. It is easy to see that the workflows on the left are bisimulation equivalent, as are the workflows on the right. Note that both the OR-Join and the AND-Join have the ANY-Join as their equivalent. Given the fundamentally different semantics of the OR-Join and the AND-Join in Standard Workflows this may come as a surprise. It can even be taken further in the sense that replacing all OR-Joins and AND-Joins in any acyclic well-behaved Standard Workflow Model with ANY-Joins as well as replacing all other constructs in Standard Workflows with their equivalent representations in Synchronizing Workflows results in an equivalent model (formally captured in Theorem 4.5). This provides a first indication of the expressive power of Synchronizing Workflows.

Definition 4.4

Let $\mathcal{W} = (\mathcal{P}, \mathcal{J}_o, \mathcal{J}_a, \mathcal{S}_o, \mathcal{S}_a, \mathcal{A}, \mathsf{Trans}, \mathsf{Name})$ *be an acyclic well-behaved Standard Workflow Model.*

The corresponding *Synchronizing Workflow Model* $\mathcal{S} = (\mathcal{P}, \mathcal{J}_o, \mathcal{J}_a, \mathcal{S}_o, \mathcal{S}_a, \mathcal{A}, \mathsf{Trans}, \mathsf{Name})$

is defined by:

$$
\begin{array}{lcll}
A^{\mathcal{S}} & = & A^{\mathcal{W}} & \#\ \textit{same activities}\ \# \\
S_o^{\mathcal{S}} & = & S_o^{\mathcal{W}} & \#\ \textit{same XOR-Splits}\ \# \\
S_a^{\mathcal{S}} & = & S_a^{\mathcal{W}} & \#\ \textit{same XOR-Splits}\ \# \\
J_o^{\mathcal{S}} & = & J_a^{\mathcal{W}} \cup J_o^{\mathcal{W}} & \#\ \textit{ANY-Joins for each of the OR-Joins \& AND-Joins}\# \\
J_a^{\mathcal{S}} & = & \varnothing & \#\ \textit{no ALL-Joins}\# \\
\mathsf{Trans}^{\mathcal{S}} & = & \mathsf{Trans}^{\mathcal{W}} & \#\ \textit{same transitions}\# \\
\mathsf{Name}^{\mathcal{S}} & = & \mathsf{Name}^{\mathcal{W}} & \#\ \textit{same labeling}\#
\end{array}
$$

□

The next step is to show that for any Standard Workflow Model the corresponding Synchronizing Workflow Model is indeed bisimulation equivalent. This is complex and requires some preparation.

First an essential property of well-behaved Standard Workflow Models is formally captured. This is the fact that in any reachable marking of a Standard Workflow net for any marked place, there is no other marked place on a path from an initial place to that marked place.

Proposition 4.2 Let $\mathcal{W}$ be an acyclic, well-behaved Standard Workflow Model, $(PN_{\mathcal{W}}, M_0)$ its corresponding net system and let $x =< p_1, t_1, p_2, t_2, ..., t_{n-1}, p_n >$ with $p_1 = p$ and $p_n = q$ be a directed path in the net $PN_{\mathcal{W}}$. For any reachable marking M we have $M(q) = 1 \Rightarrow M(p) = 0$.

Proof:

If p were marked, another token can be produced for q according to Lemma 4.6. Hence the net would not be safe. Contradiction. □

A similar result holds for Synchronizing Workflow Models, except that a distinction needs to be made between true places and false places.

Proposition 4.3 Let $\mathcal{W}$ be a Synchronizing Workflow Model, $(PN_{\mathcal{W}}, M_0)$ its corresponding net system and M a reachable marking of $(PN_{\mathcal{W}}, M_0)$. Let p be a true place and $\overline{p}$ its corresponding false place, and q another true place and $\overline{q}$ its corresponding false place such that there is a direct, acyclic path from p to either q or $\overline{q}$ then

$$(M(q) = 1 \vee M(\overline{q}) = 1) \Rightarrow (M(p) = 0 \wedge M(\overline{p}) = 0).$$

Proof:

In Synchronizing Workflow Model $\mathcal{W}$, if the place p or $\overline{p}$ contains a token, there is a firing sequence producing a token for either place q or $\overline{q}$ (Theorem 4.4). If one of these places already has a token (suppose it is a true place), according to the Monotonicity Lemma (see e.g. p.22 of [DE95]) through application of this firing sequence a second token can be produced for this place or its corresponding false place. □

Having established some basic properties of well-behaved Standard Workflows and Synchronizing Workflows, it is possible to show that any Synchronizing net can be simulated by a WB-system and vice versa, thus demonstrating that they are *simulation equivalent*. Having achieved this, it is possible to give a bisimulation relation, thus proving that they are in fact bisimulation equivalent.

The main difficulty in simulating a Standard Workflow Model by a Synchronizing Workflow Model is that the latter essentially propagates two types of tokens. For every firing of a process element of a Standard Workflow Model (propagation of a true token) we may need to fire a number of process elements in the corresponding Synchronizing Workflow Model in order to propagate some false tokens. Such a need typically arises when we want to fire an OR-Join in a Standard Workflow Model. The corresponding ANY-Join in the Synchronizing Workflow Model requires tokens for each of its inputs, hence some false tokens may need to be propagated. Lemma 4.9 guarantees that this is always possible. First however it is necessary to define the notion of workflow instances being *true-token-equivalent* which informally equates a marking of a Standard Workflow Model $\mathcal{W}$ with a marking of the corresponding Synchronizing Workflow Model $\mathcal{S}$ if for every token in the associated net of a process element of $\mathcal{W}$ there is a token in the true place of the associated net of the corresponding process element of $\mathcal{S}$.

Definition 4.5

Let $\mathcal{W}$ be a Standard Workflow Model and $\mathcal{S}$ its corresponding Synchronizing Workflow Model and let $PN_{\mathcal{W}}$ and $PN_{\mathcal{S}}$ be the corresponding Petri net systems of these models respectively. Let M_1 be a reachable marking of $PN_{\mathcal{W}}$. A reachable marking M_2 of $PN_{\mathcal{S}}$ is said to be true-token-equivalent *with M_1 iff for all places $p \in \mathsf{True}^{\mathcal{S}}, M_2(p) = 1 \iff M_1(h(p)) = 1$, where h is an injection from $\mathsf{True}^{\mathcal{S}}$ to the corresponding places in $PN_{\mathcal{W}}$, i.e. $h(rt_{x,y}) = r_{x,y}, h(ct_{x,y}) = c_{x,y}$ and $h(rt_x) = r_x$.* □

Lemma 4.9 Let $\mathcal{W}$ be a Standard Workflow Model and $\mathcal{S}$ its corresponding Synchronizing Workflow Model and let $PN_{\mathcal{W}}$ and $PN_{\mathcal{S}}$ be the corresponding Petri net systems of these models respectively. Let M_1 be a reachable marking of $PN_{\mathcal{W}}$ and M_2 a true-token equivalent marking of $PN_{\mathcal{S}}$, then there exists a (possibly empty) firing sequence $\sigma = t_1 t_2 ... t_n$ of λ-transitions in the Synchronizing Workflow Model such that $M_2 \xrightarrow{\sigma} M_2'$ where M_2' is a marking such that for every enabled OR-Join in $\mathcal{W}$ the corresponding ANY-Join in $\mathcal{S}$ is enabled.

Proof:

Without loss of generality we can focus on an enabled OR-Join with two incoming transitions in a marking M_1. As the Standard Workflow net is safe, only one ready place of the associate net of this OR-Join can hold a token (and one token only). As the marking M_2 in the corresponding Synchronizing Net is true-token-equivalent, the corresponding true place of the associated net of the ANY-Join will hold a token. According to Theorem 4.4, a token will have to arrive for the other branch of the ANY-Join. More formally, let p be the true ready place of this branch of the associated net of this ANY-Join. Then there is a marking M such that p or $\overline{p}$ is marked.

In the Standard Workflow net, according to Proposition 4.2, there cannot be a token on a path from an initial activity to the OR-Join (otherwise the OR-Join could fire twice).

As M_1 and M_2 are true-token-equivalent, in M_2 there cannot be a token in a true place on any path from an initial activity of $\mathcal{S}$ to either p or $\overline{p}$. Hence on a path from an initial activity to place p or $\overline{p}$ there must be a false place that contains a token (there can be more than one such token). Moving such tokens involves the firing of λ-transitions only (note that there cannot be ANY-Joins on such a path with true-tokens waiting, as that would mean that in the Standard Workflow net there is a token for the corresponding OR-Join in contradiction to Proposition 4.2). Note that firing these transitions will result in marking in which $\overline{p}$ is marked (p cannot be marked). □

Lemma 4.10 Let $\mathcal{W}$ be an acyclic well-behaved Standard Workflow Model, $(PN_{\mathcal{W}}, M_0)$ its corresponding net system, $\mathcal{S}$ its corresponding Synchronizing Workflow Model and $(PN_{\mathcal{S}}, M_0)$ its net system, then $PN_{\mathcal{W}}$ and $PN_{\mathcal{S}}$ are simulation equivalent.

Proof:

First focus is on simulating the Synchronizing Workflow Model by the Standard Workflow Model. This is achieved through induction on the number n of firings of process elements. The case of $n = 0$ follows from the fact that the initial markings of these workflow models are true-token-equivalent. Assume that after firing n process elements, marking M_1 of the Synchronizing Workflow Net and marking M_2 of the Standard Workflow Net are true-token-equivalent. We then fire a process element p in the Synchronizing Workflow Model which results in marking M_1'.

We will show that either M_1' and M_2 are true-token-equivalent or it is possible to fire a corresponding element of the Standard Workflow Model and the resulting marking M_2' and M_2 are true-token-equivalent.

This is achieved through a straightforward case distinction:

1. If p was enabled with only false tokens, firing it resulted in a marking that is true-token-equivalent to M_2. No action needs to be performed in the Standard Workflow Model. From this moment on we assume that at least one of the enabling tokens of p was a true token.
2. If p is an activity, the corresponding activity in the Standard Workflow Model can be performed.
3. If p is a Split, the corresponding Split in the Standard Workflow Model can be performed, and the resulting marking is again true-token-equivalent.
4. If p is an ANY-Join with more than one true token, one can conclude that the corresponding Join in the Standard Workflow Model has to be an AND-Join, as otherwise this workflow would not be safe. Firing this AND-Join results again in true-token-equivalent markings.
5. If p is an ANY-Join with one true and the rest false tokens, one can conclude that the corresponding Join in the Standard Workflow Model has to be an OR-Join, as otherwise there would be a deadlock (as according to Proposition 4.3 there are no tokens in places above). Firing this OR-Join results again in true-token-equivalent markings.

The opposite, simulating the Standard Workflow Model by the corresponding Synchronizing Workflow Model, uses a similar case distinction. The only real problem is that Lemma 4.9 is needed in order to guarantee that if an OR-Join is performed in the Standard Workflow net, the corresponding ANY-Join in the Synchronizing Workflow Model can be performed. □

Corollary 4.3 Let $\mathcal{W}$ be an acyclic well-behaved Standard Workflow Model, $PN_{\mathcal{W}}$ its corresponding net, $\mathcal{S}$ its corresponding Synchronizing Workflow Model and $PN_{\mathcal{S}}$ its net, then for every reachable marking M of $PN_{\mathcal{W}}$ there is a reachable marking M' of $PN_{\mathcal{S}}$ which is true-token-equivalent to M. Similarly for every reachable marking M of $PN_{\mathcal{S}}$ there is a reachable marking M' of $PN_{\mathcal{W}}$ true-token-equivalent to M.

Theorem 4.5 Let $\mathcal{W}$ be an acyclic well-behaved Standard Workflow Model, $PN_{\mathcal{W}}$ its corresponding net, $\mathcal{S}$ its corresponding Synchronizing Workflow Model and $PN_{\mathcal{S}}$ its net, then $PN_{\mathcal{W}}$ and $PN_{\mathcal{S}}$ are bisimulation equivalent.

Proof:

Before defining a bisimulation relation, we introduce labels for all transitions, except those that just propagate false tokens. The reason for this is that we would like the bisimulation relation to maintain the relationship between the execution of the various corresponding joins and splits in the two nets. Naturally, by showing that the resulting nets are equivalent, it follows that the original nets with fewer labels, are also equivalent. The bisimulation relation is just made a bit more strict.

We now define a relation R between the reachable markings of both nets and show that it is a bisimulation relation. Formally, R relates two markings if and only if they are true-token-equivalent. From Corollary 4.3 it then follows that for every reachable marking M_1 of the Standard Workflow net, there is a reachable marking M_2 of the Synchronizing Workflow net such that $(M_1, M_2) \in R$ and vice versa.

Let $(M_1, M_2) \in R$. First it will be shown that for every label a and marking M_1' of the Standard Workflow net such that $M_1 \stackrel{a}{\Longrightarrow} M_1'$ there is a marking M_2' in the Synchronizing Workflow net with $M_2 \stackrel{a}{\Longrightarrow} M_2'$ and $(M_1', M_2') \in R$. This requires the following case distinction:

1. If the label corresponds to an activity, then the corresponding activity can be performed in the Synchronizing Workflow net.
2. If the label corresponds to a split, then the corresponding split can be performed in the Synchronizing Workflow net.
3. If the label corresponds to an AND-Join then the corresponding join in the Synchronizing Workflow net can be performed, as all input branches will have true tokens.
4. The case where the label corresponds to an OR-Join is the most interesting one. In the Synchronizing Workflow net, the corresponding join will have exactly one true token. If all other branches have false tokens, then the join can be performed directly. If not, then according to Lemma 4.9, false tokens can be propagated using λ-transitions only.

Note that in all the above cases, the resulting markings are true-token-equivalent to the resulting marking in the Standard Workflow net, hence related in R.

Now it will be shown that for every label a and marking M_2' of the Synchronizing Workflow net such that $M_2 \stackrel{a}{\Longrightarrow} M_2'$ there is a marking M_1' in the Standard Workflow net with $M_1 \stackrel{a}{\Longrightarrow} M_1'$ and $(M_1', M_2') \in R$. This requires a similar case distinction (note that we do not need to consider transitions propagating false tokens, as such transitions are unlabelled):

1. If the label corresponds to an activity, then the corresponding activity can be performed in the Standard Workflow net.
2. If the label corresponds to a split, then the corresponding split can be performed in the Standard Workflow net.
3. If the label corresponds to an ANY-Join with more than one true token, then this join corresponds to an AND-Join in the Standard Workflow net (otherwise the workflow would not be safe). This AND-Join can be performed as all its input branches will have tokens.
4. If the label corresponds to an ANY-Join with one true token and the rest false tokens, then this join corresponds to an OR-Join in the Standard Workflow net (otherwise deadlock would occur). Again, this OR-Join can then fire.

Note that in all the above cases the resulting markings are true-token-equivalent to the resulting marking in the Synchronizing Workflow net, hence related in R. Therefore, and given that R relates the initial markings of both systems, R is a bisimulation relation.

□

Theorem 4.5 has important practical ramifications, as it effectively demonstrates that the choice for a true/false token evaluation strategy when developing a workflow engine does not compromise the expressive power of the workflow language involved as long as well-behaved workflows with structured loops only are considered. One advantage of this approach is that workflow analysts need not worry about deadlock, as all their specifications are guaranteed to be deadlock free.

Having established which Standard Workflow Models can be captured as Synchronizing Workflow Models, one may wonder whether all Synchronizing Workflow Models have a Standard Workflow equivalent. Intuitively, the fact that the Petri net representation of both ANY-Join and ALL-Join is non-free-choice hints at the possibility that Synchronizing Workflow Models may exist which do *not* have a Standard Workflow equivalent. The next theorem proves this fact formally.

Theorem 4.6 There exist Synchronizing Workflow Models for which no Standard Workflow Model can be found such that the corresponding Petri nets are bisimulation equivalent.

Proof:

Consider the Synchronizing Workflow Model $\mathcal{W}$ of Figure 32. We will show that no free-choice net system exists that is equivalent to this workflow. This then concludes the proof as the corresponding net system of any Standard Workflow Model is free-choice.

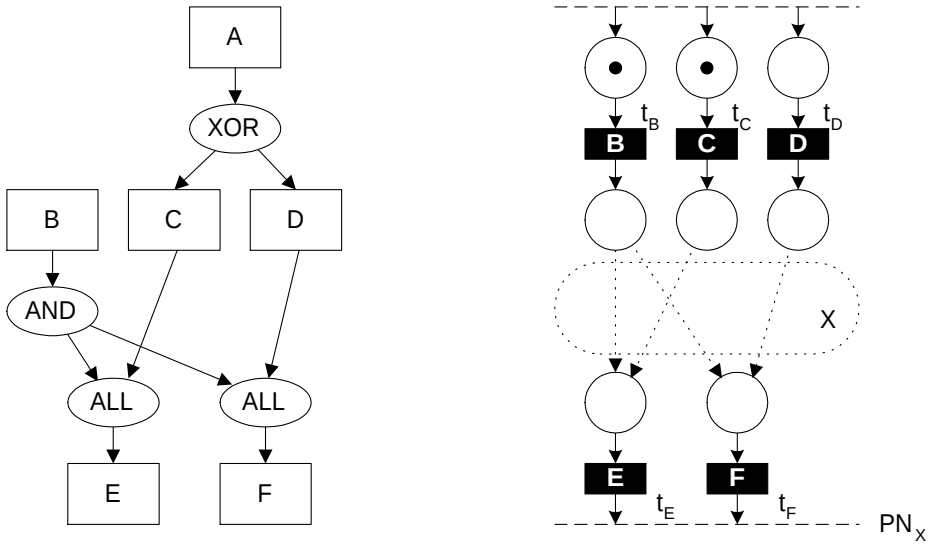

Figure 32: ALL-Join adds expressive power

Let $PN_{\mathcal{W}}$ be the corresponding net of Synchronizing Workflow Model $\mathcal{W}$. Let $\mathcal{X}$ be a Standard Workflow Model that is equivalent to $\mathcal{W}$ and PN_x its corresponding net.

When establishing a bisimulation relation, we have that in PN_x there must be a reachable marking M_1 that enables a transition labelled A and a transition labelled B and does not enable any transitions labelled C, D, E or F. Let us refer to the enabled transitions as t_A and t_B respectively.

For PN_x to be bisimulation equivalent to $PN_{\mathcal{W}}$ there must be a marking M_2 such that $M_1 \stackrel{a}{\Longrightarrow} M_2$ and M_2 is a marking that enables t_B and a transition labelled C and does not enable any transitions labelled A, D, E or F. Let us refer to the enabled transition labelled C in M_2 as t_C.

Similarly must be a marking M_3 such that $M_1 \stackrel{a}{\Longrightarrow} M_3$ and M_3 is a marking that enables t_B and a transition labelled D and does not enable any transitions labelled A, C, E or F. Let us refer to the enabled transition labelled D in M_3 as t_D.

The bisimulation construction further yields that there must be a marking M_4 such that $M_2 \stackrel{bc}{\Longrightarrow} M_4$ and M_4 is a marking that enables a transition labelled E and does not enable any other labelled transitions. Let us refer to the enabled transition labelled E in M_4 as t_E.

Similarly there must be a marking M_5 such that $M_2 \stackrel{bd}{\Longrightarrow} M_5$ and M_5 is a marking that enables a transition labelled F and does not enable any other labelled transitions. Let us refer to the enabled transition labelled F in M_5 as t_F.

As PN_x is the corresponding net of a Standard Workflow Model, transitions t_B, t_C, t_D, t_E and t_F have exactly one input and one output place. A subnet of PN_x along with transitions t_B, t_C, t_D, t_E and t_F with marking M_2 is schematically shown as the right diagram of Figure 32. The subnet comprising output places of transitions t_B, t_C and t_D, input places of transitions t_E, t_F and subnet X of Figure 32 fulfils the assumptions of

Lemma 4.3 (Selective Synchronizer) hence PN_x cannot be free-choice which contradicts the assumption that PN_x is the corresponding net of a Standard Workflow Model.

□

Summarizing, as opposed to Standard Workflow Models, Synchronizing Workflow Models are always safe, they never result in deadlock, and they do not allow for direct specification of arbitrary cycles [ABHK00, AHKB00, AHKB02, WPH02]. Synchronizing Workflow Models can express all Standard Workflow Models that do have these properties (i.e. well-behaved, acyclic models). There are Synchronizing Workflow Models though that are inherently non free-choice and hence do not have a Standard Workflow equivalent.

5 Advanced Expressiveness Results

The different evaluation strategies and the semantics of the basic control flow constructs are not the only areas not precisely addressed by the WfMC. In this section we would like to investigate some other issues associated with choices that workflow engine designers are likely to face.

When defining syntax and semantics for workflow models, it was assumed that there may be multiple final activities in a workflow model. There are some workflow engines (e.g. Verve Workflow, Forté Conductor, etc.) for which this assumption does not hold. In Section 5.1 the consequences associated with this design decision are explored.

The execution of Standard Workflow Models (as opposed to Synchronizing Workflow Models) may result in deadlock. Typically this is viewed as an undesirable situation. In Section 5.2 we consider the possibility of using deadlock intentionally to express certain task dependencies and determine whether this can enhance the expressive power of Standard Workflow Models.

Some workflow languages support constructs, not part of the basic control flow constructs, which clearly have practical significance. One such construct is considered in Section 5.3, where it is shown that it cannot be simulated using the basic control flow constructs.

5.1 Termination

Termination refers to the state where no work remains to be done. Often, this situation is referred to as successful termination to distinguish it from deadlock [BW90]. While the presented definition of termination in Section 2.1 seems straightforward, and languages supporting the synchronizing evaluation strategy employ it (e.g. MQSeries Workflow), most workflow engines in practice, especially those supporting standard or safe workflows (a notable exception here is Staffware), have a different view on termination. In these engines, for every workflow, one or more final tasks need to be specified. The workflow then is considered to be terminated when the first of these final tasks has completed.

This termination policy is particularly problematic when a (or the) final task is reached while some other parallel threads are still running. What the workflow engine will do in such a situation differs from product to product but typically the remaining threads are abruptly

aborted leaving the workflow in a potentially inconsistent state, i.e., a state where the instance is blocked or its behaviour is unspecified. Hence we are interested in workflows where this situation cannot occur and we will refer to them as terminating *strictly*.

Definition 5.1 (*strictly terminating workflows*)
Let $(PN_{\mathcal{W}}, M_0)$ *be the corresponding system of Standard Workflow Model* $\mathcal{W}$*. We will call* $\mathcal{W}$ terminating strictly *iff for every sink transition* t *and every reachable marking* M *of* $PN_{\mathcal{W}}$ *that enables* t*, we have for all places* p*:*

$$M(p) = \begin{cases} 1 & \text{if } p \in {\bullet t} \\ 0 & \text{if } p \notin {\bullet t} \end{cases}$$

□

Definition 5.2 (*uniquely terminating workflows*)
A Standard Workflow Model $\mathcal{W}$ *is* terminating uniquely *iff it has exactly one final task and is terminating strictly.* □

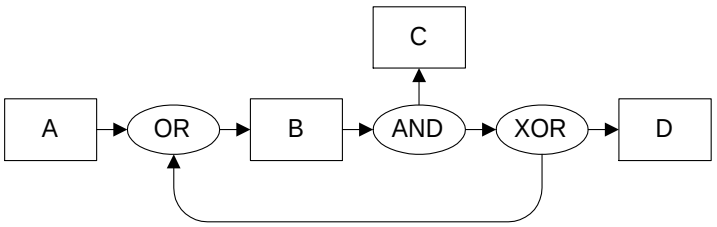

Figure 33: Sample Standard Workflow Model utilising relaxed termination policy

Clearly, there exist non-safe Standard Workflow models for which there is no strictly terminating equivalent workflow model. A simple example of such a model is shown in Figure 33 and the practical usefulness of a relaxed termination strategy is evident when considering patterns involving multiple instances (see [ABHK00, AHKB00, AHKB02]). Standard Workflow models that are well-behaved, on the other hand, have a terminating uniquely equivalent workflow model.

To prove this result we will show that for every well-behaved Standard Workflow model $\mathcal{W}$, it is possible to transform its corresponding Petri net, PN into a bisimulation equivalent net PN' that has only one sink transition (i.e. a transition without output places). It is then possible to convert PN' back to a terminating uniquely workflow specification that is equivalent to $\mathcal{W}$.

Theorem 5.1 Every well-behaved Standard Workflow Model has an equivalent workflow model that is terminating uniquely.

Proof:
Let $\mathcal{W}$ be a well-behaved Standard Workflow model and $PN = (P, T, F)$ be its corresponding WB-net with S_{PN}, P^s and T^s as defined in Definition 4.3. Then,

$$\begin{aligned} PN' \quad = \quad & (P \cup \{p_s | s \in S_{PN}\}, \\ & T \cup \{t_f\}, \\ & F \cup \{(t, p_s) | s \in S_{PN} \wedge t \in T^s\} \cup \{(p_s, t_f) | s \in S_{PN}\}) \end{aligned}$$

is a WB-net with one sink transition t_f.

Clearly, t_f is a sink transition. There are no other sink transitions because all (former) sink transitions in S_{PN} have an output place in $\{p_s | s \in S_{PN}\}$ (Note that $s \in T^s$ and $s\bullet = \{p_s\}$). Moreover, the source places of PN are still the only source places of PN', PN' is free-choice, and has no sink places (all new places have an input and output transition). It is also easy to see that every node is on a path from a source place. To prove the last three properties stated in Proposition 4.1, we show that from any reachable state it is possible to reach the empty state, i.e., enable and fire t_f.

Consider a (former) sink transition $s \in S_{PN}$. As long as P^s contains tokens, there is a firing sequence enabling s (Lemma 4.6). If a transition in T^s fires, the last token in P^s is consumed. Moreover, s can fire only once (Lemma 4.7). Note that $s \in T^s$ and exactly one source place is in P^s. On the one hand, only one transition of T^s can fire. Therefore, it is not possible to mark p_s more than once. On the other hand, at least one of the transitions of T^s will fire (assuming fairness: initially the unique source place in P^s is marked and it is possible to reach the empty marking). Therefore, p_s will be marked at least once. Hence, the place p_s is marked once. Therefore, all places in $\{p_s | s \in S_{PN}\}$ are marked once and sink transition t_f will produce the empty marking.

As the resulting PN' net is an FCDA-net (as per Definition 4.1) it is straightforward to transform PN' back to a workflow model using transformations presented in Theorem 4.2.

It remains to be shown that PN' is bisimulation equivalent to PN. Let M be a reachable marking of PN. We will call a reachable marking M' of PN' an associated marking of M iff $M'[P] = M$ (in other words it should have exactly the same number of tokens in every place of the original net, the markings of the introduced places does not matter). From the construction of PN' it is easy to check that a relation R that relates every marking M of PN to all its associated markings in PN' is indeed a bisimulation relation.

□

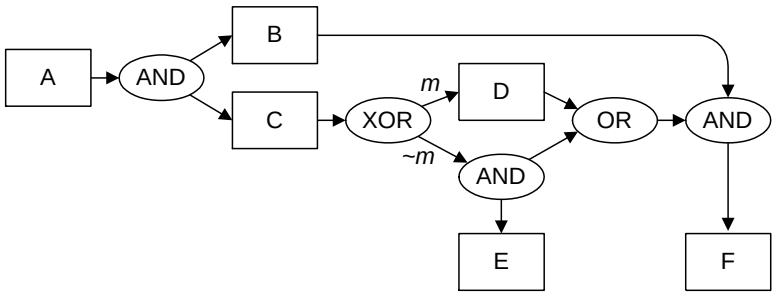

Figure 34: Sample Standard Workflow model with two final tasks

Example 5.1 As an example of the construction used in the proof of Theorem 5.1, consider the workflow of Figure 34. This workflow has two final tasks, named E and F. In every instance, the task named F is executed while the task named E is executed only if condition m evaluates to false. By following the construction presented in the proof we end up with the workflow presented in Figure 35. Note that this workflow is indeed

equivalent to the one of Figure 34. Also note that from a comprehensibility point of view, the workflow with the unique final task is much more complicated and its control flow would be much harder to understand for a workflow designer. □

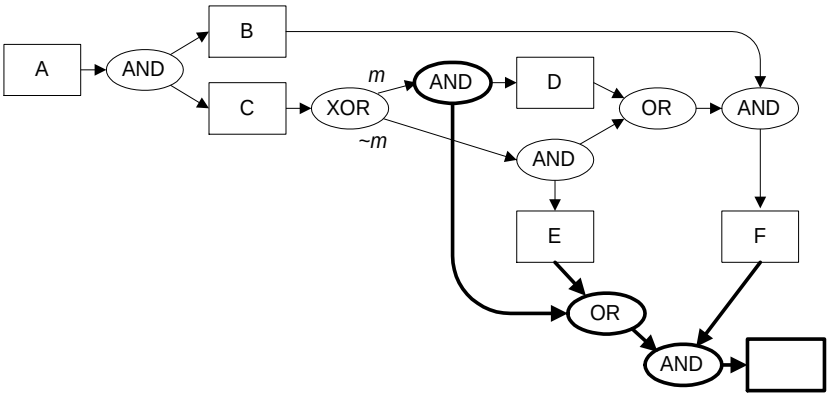

Figure 35: Terminating uniquely equivalent workflow to workflow of Figure 34

Remark 5.1

Naturally, the equivalent of Theorem 5.1 for Synchronizing Workflow models is trivial, as for every Synchronizing Workflow Model $\mathcal{W}$ with more than one final task, the Synchronizing Workflow Model which simply adds an ANY-Join with input transitions from all the final tasks followed by a null activity is equivalent to $\mathcal{W}$. □

5.2 Deadlock

This section takes a closer look at the issue of deadlock in workflows. As Synchronizing Workflow models cannot deadlock, focus is on Standard Workflows exclusively.

Imagine a workflow management system that has the ability to detect deadlock at runtime (from a programming point of view this is fairly easy to achieve). Moreover, imagine that the workflow analyst could instruct the workflow engine what to do when it encounters a deadlock. Specifically, (s)he could instruct the engine to treat deadlock as a normal, successful, termination[7]. The question that we would like to address is whether such a feature would increase the expressive power of a workflow engine. More formally, this question boils down to determining whether any Standard Workflow model with a deadlock has an "equivalent" deadlock free Standard Workflow model. As our equivalence notion will always distinguish a specification that deadlocks from a specification that does not deadlock, a relaxed equivalence notion is required.

Definition 5.3

Workflow models $\mathcal{W}_1$ and $\mathcal{W}_2$ are execution equivalent *iff the begin-end transformations of their corresponding systems are bisimilar according to Definition 3.4 excluding the second clause.* □

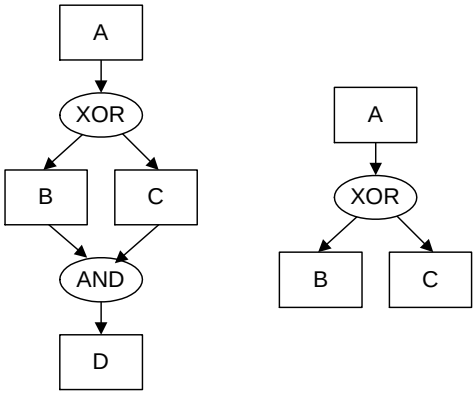

Figure 36: Two execution equivalent processes

Example 5.2 The two workflow processes depicted in Figure 36 are execution equivalent even though the left-most process deadlocks whilst the right-most process always terminates successfully. □

Theorem 5.2 (*dynamic deadlock resolution adds expressive power*) There exist Standard Workflow models for which no deadlock free execution equivalent Standard Workflow model exists.

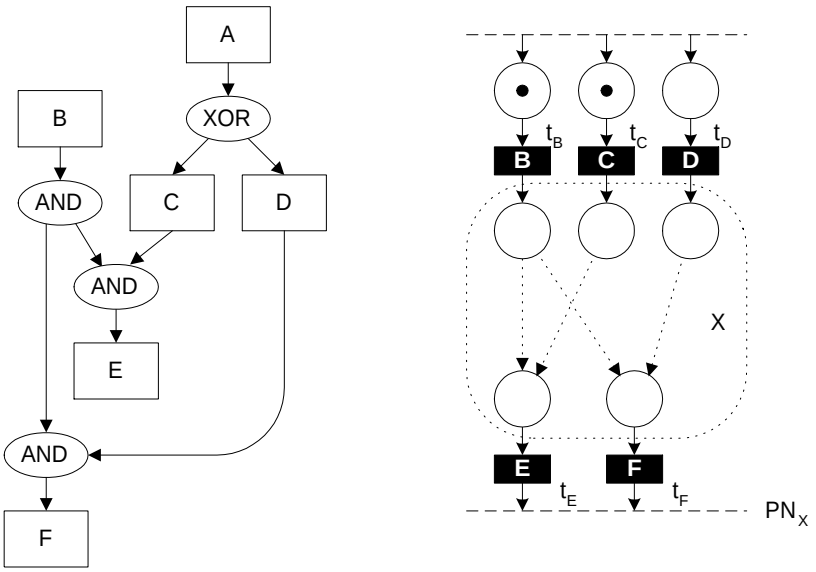

Figure 37: Standard Workflow Model with a deadlock

Proof:

Consider the Standard Workflow model $\mathcal{W}$ of Figure 37. The semantics of this workflow

[7] We are not aware of any commercial workflow engine with this capability.

specification is as follows. After completing activity A a choice is made between activities C and D. At the same time activity B can be performed. If C is chosen and completed along with B, activity E can be performed. If D is chosen and completed along with B, activity F can be performed.

The rest of the proof is analogous to the proof of Theorem 4.6. Using the same argumentation we have that in any net PN_x that is bisimulation equivalent to $\mathcal{W}$ there must be transitions labelled B, C, D, E and F (let us call these transitions t_B, t_C, t_D, t_E and t_F). Furthermore if M_1 is a reachable marking of PN_x such that it enables transitions t_B and t_C and no other labelled transitions there must be a firing sequence σ_1 such that $M_1 \stackrel{\sigma_1}{\Longrightarrow} M_2$ and M_2 is a marking that enables transition t_E and no other labelled transition. Similarly if M_3 is a reachable marking of PN_x such that it enables transitions t_B and t_D and no other labelled transitions there must be a firing sequence σ_2 such that $M_3 \stackrel{\sigma_2}{\Longrightarrow} M_4$ and M_4 is a marking that enables transition t_F and no other labelled transition (this is shown in the right diagram of Figure 37). The subnet X of this diagram fulfils the conditions of Lemma 4.3 (Selective Synchronizer) and we have that PN_x cannot be free-choice or it deadlocks.

□

Theorem 5.2 may strike the reader as controversial as deadlock in a specification would always seem to be undesirable. However, the theorem shows that from an expressiveness point of view it is advantageous to be able to instruct a workflow engine what to do in case it encounters a deadlock at runtime. If this option were present in the engine, deadlock could be used as a constructive tool to help design processes that otherwise can not be specified.

5.3 Advanced Synchronization

Standard Workflow models support two types of merge constructs: the AND-Join and the OR-Join. There exist business patterns though that are hard or impossible to capture using these types of merges only. An example of such a pattern is the *discriminator* described in [ABHK00, AHKB00, AHKB02, WPH02].[8]

The discriminator is a merge construct with a fairly straightforward intuitive semantics. It behaves like an OR-Join in the sense that it is nonsynchronizing, an incoming branch can fire the activity following the discriminator, but it is different in the sense that the subsequent activity should not be fired by every incoming branch, only by the one that finishes first.

Figure 38 shows a very basic process model using the discriminator construct. In this model, from the initial marking enabling activities A and B the following scenarios are possible:

1. Activity A is completed. Activity C gets enabled and the process finishes when both activities B and C are completed.

2. Activity B is completed. Activity C gets enabled and the process finishes when both activities A and C are completed.

[8] The term discriminator has been adopted from Verve [Ver00] and is also referred to as partial join [Cas98, CPP98] or 1-out-of-N join.

3. Activities A and B are completed before activity C is started. The process finishes once activity C completes. Note that activity C is enabled as soon as either A or B completes.

The important feature of the discriminator in this model is that activity C can be done only once. Formally this behaviour can be captured by the Petri net system PN_D in Figure 38 (note that this net system is not free-choice).

The following theorem shows that the discriminator adds expressive power to Standard Workflow Models, as it is inherently non free-choice. The proof of the theorem was inspired by the results in [Smi96].

Theorem 5.3 (*the discriminator adds expressive power*) There is no Standard Workflow Model equivalent to the Petri net system PN_D of Figure 38.

Proof:

Suppose that there is a deadlock-free, free-choice Petri net that is bisimulation equivalent

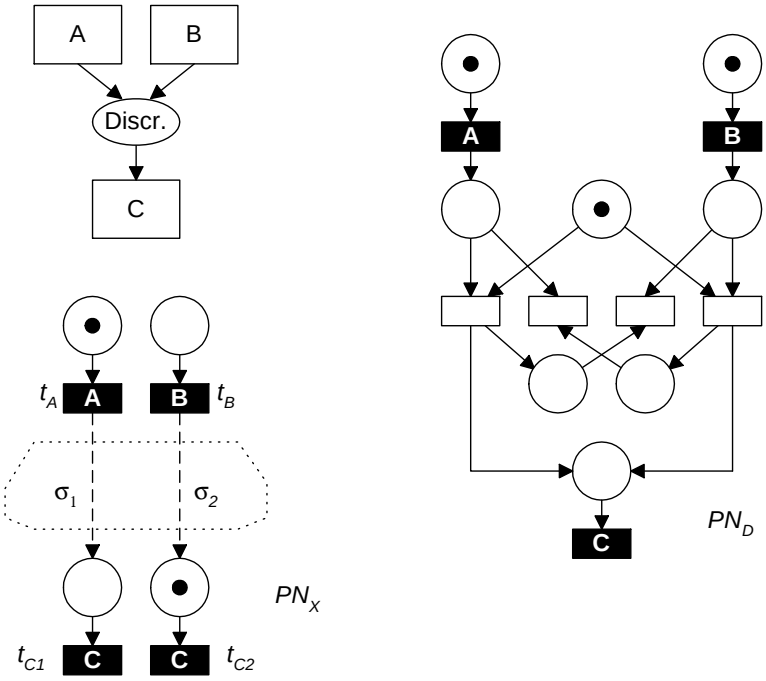

Figure 38: Illustration of the discriminator proof

to some Standard Workflow Model $\mathcal{W}$. Let us refer to this net as $\mathcal{S}$. This net has to have a transition labelled A and a transition labelled B. We will call these transitions t_A and t_B respectively.

Let M_{AB} be a reachable marking of $\mathcal{S}$ that enables transitions t_A and t_B. When establishing a bisimulation relation, it turns out that there must be a firing sequence σ_1 such that $M_{AB} \stackrel{t_A\sigma_1}{\longrightarrow} M_{BC}$ and M_{BC} is a marking of $\mathcal{S}$ that enables transition t_B and a

transition labelled C (let us call it t_{C1}) but does not enable t_A (or any other transition labelled with A).

Similarly there must be firing sequence σ_2 such that $M_{AB} \xrightarrow{t_B\sigma_2} M_{AC}$ and M_{AC} is a marking of $\mathcal{S}$ that enables transition t_A and a transition labelled C (let us call it t_{C2}) but does not enable t_B.

Consider now the simulation scenario in which from marking M_{AB} of $\mathcal{S}$, the firing sequence $t_B\sigma_2$ is performed resulting in marking M_{AC} (see the marking of PN_X shown in Figure 38). If it was possible from marking M_{AC} to perform the firing sequence $t_A\sigma_1$, it would be possible to fire both transitions t_{C1} and t_{C2} (if $t_{C1} = t_{C2}$ then it would be possible to fire that transition twice). As that would make $\mathcal{S}$ not equivalent to $\mathcal{W}$, consider the first transition in σ_1 that cannot be fired. If it is possible to enable it by firing some other non-labelled transitions, consider the next transition in σ_1 for which this is impossible. Let us refer to this transition as t_q. This transition must have at least one token in one of its input places. But as $\mathcal{S}$ is free-choice, any other transition that shares its input places with t_q must share all its input places with t_q and therefore it cannot be fired either. As there is no possibility to remove the token from one of the input places of t_q, there is a firing sequence from the marking M_{AC} that results $\mathcal{S}$ to be in deadlock and hence it is not equivalent to $\mathcal{W}$.

□

Considering the semantics of the discriminator in the more general case raises the question as to how it should behave in loops. The simplest solution would be to allow the first incoming branch to trigger the activity following the discriminator and ignore all the other branches from then on. Clearly though this causes a deadlock when the discriminator is used in a loop. A more sophisticated approach would be to allow the first incoming branch to trigger the activity following the discriminator, and to keep track of the other branches. Once all branches have completed, the discriminator is "reset" and the next incoming branch to finish can again trigger it. This semantics is captured formally by the Petri net shown in Figure 39.

In Figure 39 two activities A and B are shown, which are input to a discriminator d (the schema extends in a natural way to the case of n incoming branches). The place named $Start_d$ initially contains a token (this place is a status place as used in Definition 5.6). This represents the situation that the discriminator is waiting for one of its incoming branches to finish. When the first incoming branch finishes, say activity A, activity X is enabled, a token is produced for the place W_d^B to represent the fact that the discriminator still needs to wait for activity B before it can be reset, and a token is placed in place S_d^A so that the fact is remembered that the branch with activity A was already "seen". The completion of B now does not lead to another instance of activity X, rather a token is removed from W_d^B and put in S_d^B. As both branches have now been executed, tokens can be removed from S_d^A and S_d^B and a token can be put in $Start_d$, representing the fact that the discriminator is reset and ready for another iteration. Note that this semantics works well for models that are not guaranteed to be safe, for example the completion of two instances of activity A before an instance of activity B is completed, simply results in the first instance enabling activity X, and the second instance having to wait for an instance of B before it can enable activity X again.

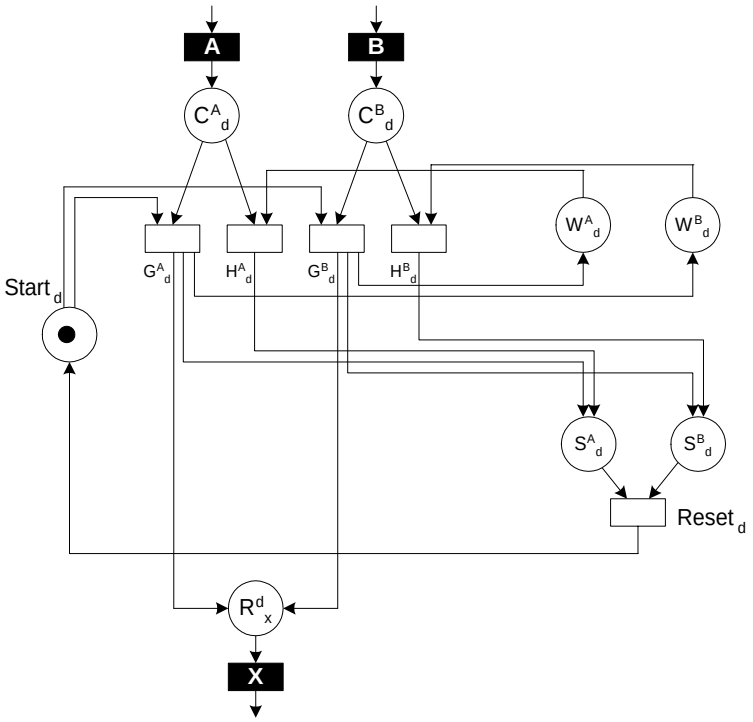

Figure 39: Petri net semantics of the discriminator

Definition 5.4

Syntactically, a Standard Workflow model with discriminators, is a Standard Workflow model $\mathcal{W}$ with a nonempty set $\mathcal{D}$ of discriminators. Each discriminator has an indegree of at least two and an outdegree of one. □

Definition 5.5

Given a Standard Workflow model $\mathcal{W}$ with discriminators from $\mathcal{D}$, the corresponding, marked, labelled, Petri system $PN_{\mathcal{W}} = (P'_{\mathcal{W}}, T'_{\mathcal{W}}, F'_{\mathcal{W}}, L'_{\mathcal{W}}, M'_{\mathcal{W}})$ is defined by:

$$
\begin{aligned}
P'_{\mathcal{W}} &= P_{\mathcal{W}} \cup \\
&\quad \{w_d^x \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup && \#\text{“waiting” places}\# \\
&\quad \{s_d^x \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup && \#\textit{branches already seen}\# \\
&\quad \{Start_d \mid d \in \mathcal{D}\} && \#\text{“start” places}\#
\end{aligned}
$$

$$
\begin{aligned}
T'_{\mathcal{W}} &= T_{\mathcal{W}} \cup \\
&\quad \{G_d^x \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup && \#\textit{transitions to trigger discriminator}\# \\
&\quad \{H_d^x \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup && \#\textit{transitions not to trigger discriminator}\# \\
&\quad \{Reset_d \mid d \in \mathcal{D}\} && \#\text{“reset” transitions}\#
\end{aligned}
$$

$$L'_{\mathcal{W}} = L_{\mathcal{W}} \cup \{(t, \lambda) \mid t \in T'_{\mathcal{W}} \setminus T_{\mathcal{W}}\}$$

$$F'_{\mathcal{W}} = F_{\mathcal{W}} \cup$$

$\{(Start_d, G_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(Reset_d, Start_d) \mid d \in \mathcal{D}\} \cup$
$\{(c_d^x, G_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(c_d^x, H_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(G_d^x, w_d^y) \mid y \neq x \wedge d \in \mathcal{D} \wedge y \in in(d) \wedge x \in in(d)\} \cup$
$\{(G_d^x, s_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(G_d^y, r_x^d) \mid d \in \mathcal{D} \wedge y \in in(d) \wedge x \in out(d)\} \cup$
$\{(w_d^x, H_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(H_d^x, s_d^x) \mid d \in \mathcal{D} \wedge x \in in(d)\} \cup$
$\{(s_d^x, Reset_d) \mid d \in \mathcal{D} \wedge x \in in(d)\}$

The initial marking $M'_{\mathcal{W}}$ assigns a single token to each of the places in $\{r_x \mid x \in \mathcal{I}\}$ and to each of the places in $\{Start_d \mid d \in \mathcal{D}\}$. □

Definition 5.4 raises an interesting question regarding the termination of a workflow model containing a discriminator. According to Definition 2.11, any workflow containing a discriminator will never terminate as the token in place Start_d cannot be removed.

When faced with constructs utilizing tokens that keep track of the state of these constructs, rather than the state of the process, the definition of termination needs to be adapted.

Definition 5.6 (*relaxed termination for advanced workflows*)
Refer to places that contain tokens in the initial marking of the corresponding Petri net of some workflow specification, but do not correspond to initial places of workflow elements, as status places*. The workflow specification can terminate iff from the initial marking of its corresponding Petri net system a marking can be reached, where only status places contain tokens.* □

It is possible to assign a meaningful semantics to the concept of a discriminator for synchronizing languages. However, there are multiple choices. One could define the discriminator such that it passes on the first token that it receives and ignores tokens from the other activities (till every such activity has generated a token in which case the next cycle could start), or it could be defined in such a way that it passes on the first true-token and waits for tokens from the other activities, but generates a false-token when it receives false-tokens from each of its input activities. Other interpretations are possible as well, but as to the best of our knowledge there is no commercially available workflow system that uses a synchronizing strategy and provides support for the discriminator, this issue will not be explored further.

5.4 Summary

In this section we have presented several issues that focus on more advanced aspects of workflow specification beyond the use of standard control flow constructs such as AND-Joins, AND-Splits, OR-Joins and OR-Splits. In Section 5.1 we argued against a *strict* termination policy which is commonly deployed by workflow vendors. Additionally we presented a transformation that can be used to transform well-behaved models that have more than one final task into

models with one final task. This transformation is useful to any workflow modeller working with a language that requires a unique final task in a workflow process. In Section 5.2 we presented theoretical arguments for adopting an *active deadlock resolution* strategy. We are not aware of any workflow language that employs such a strategy. In Section 5.3 we provided a proof that a *Discriminator* construct is not possible to implement using Standard Workflow Models. We consider it to be a good argument for adoption of the discriminator construct (or the more general partial join [Cas98, CPP98]) in a modern workflow modelling language.

6 Conclusions

In this paper the focus was on expressiveness results for workflow languages as far as their support for control flow is concerned. The main results are summarized in Figure 40. In this figure, all the arrows represent strict inclusion relations.

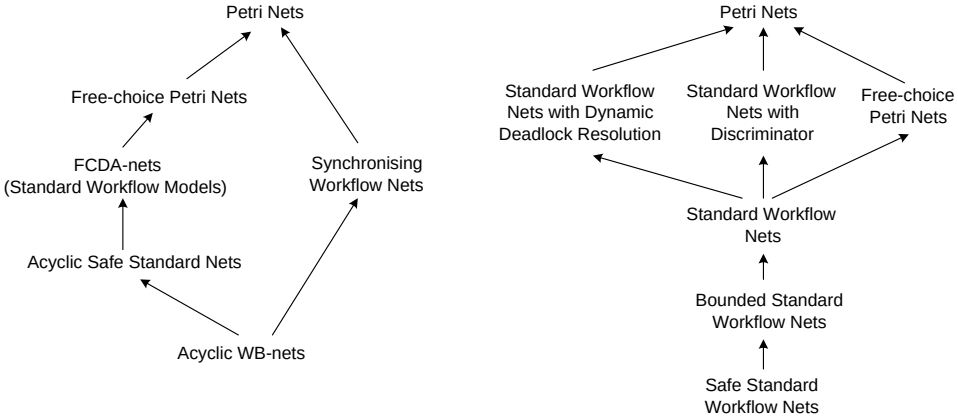

Figure 40: Summary of expressiveness results

Table 1 provides a comparison of some commercially available workflow systems (WFMSs) in terms of some of the features discussed in this paper. In this table, SAP R/3 Workflow and Filenet's Visual Workflow's evaluation strategies are termed "Restricted Safe" as the syntactic restrictions imposed by these products guarantees that workflows are safe. Deadlock can occur in specifications of some WFMSs, as indicated. No WFMS supports dynamic deadlock resolution though. Custom Joins use the data perspective to achieve more advanced forms of routing and, like the discriminator, may result in behaviour that cannot be captured by free-choice constructs. For a more detailed evaluation of a larger set of contemporary workflow systems and more information on the versions evaluated, we refer to [Kie02, AHKB02, WPH02].

It is our hope that the results in this paper will aid both workflow analysts and workflow engine designers. For workflow analysts the results, among others, will allow them to understand the inherent limitations of the languages they need to specify their workflows in. For workflow engine designers the results suggest directions for improving the expressive power of their engine.

In this paper focus was on the control flow perspective only. We believe that it is important that control flow and data flow are separated as much as possible, as workflows become harder

product	Features				
	Evaluation Strategy	Termination	Deadlock	Arbitrary Loops	Advanced Synch.
MQ Series/Workflow	Synch.	Relaxed	Never	No	-
Visual Workflow	Restr. Safe	Strict	Never	No	-
Forte Conductor	Standard	Strict	Can	Yes	Custom Join
Verve	Standard	Strict	Can	Yes	Discriminator
InConcert	Synch.	Relaxed	Never	No	-
SAP R/3 Workfow	Restr. Safe	Strict	Never	No	Custom Join
Staffware	Safe	Relaxed	Can	Yes	-
I-Flow	Safe	Strict	Can	Yes	-
HP ChangEngine	Safe	Strict	Can	Yes	Custom Join

Table 1: Comparison of Features for some WFMSs

to (formally) analyse and understand, the moment part of their control flow is "hidden" in the data flow. Hence, it is imperative to first understand expressiveness issues within the control flow perspective before considering data flow. Nevertheless, the inclusion of data flow and its implications for expressiveness are considered an important avenue for further research. Another topic for future research are transactional aspects. Note that at the lower levels mechanisms such as a two-phase commit are used to synchronize various parts of the workflow. It could also be the case that an activity may start once another activity is started (i.e., before completion). We did not consider such dependencies in this paper because they seem at another level of granularity and are not supported by the current generation of workflow products.

Acknowledgments

We would like to thank Javier Esparza for providing us with the reference to Einar Smith's paper, Eric Verbeek for his comments on an early draft of this paper, and the two anonymous reviewers for providing many constructive comments.

A Petri Nets: Notations and Definitions

This section introduces basic Petri net terminology and notations and is adapted from [DE95]. Readers familiar with Petri nets can skip this section. For more information on basic Petri net theory, see [Mur89, RR98].

The classical Petri net is a directed bipartite graph with two node types called *places* (graphically represented by circles) and *transitions* (graphically represented by thick lines). The nodes are connected via directed *arcs*.

A *Petri net* is a tuple $PN = (P, T, F)$ where P and T are finite disjoint sets of *places* and *transitions* respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation).

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We

use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state* M, often referred to as marking, is the distribution of tokens over places, i.e., M is a function mapping the set of places P onto the natural numbers: $M \in \mathbb{N}^P$. We will represent a marking as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the marking with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this marking as follows: $p_1 + 2p_2 + p_3$. If confusion is possible, we use brackets to denote markings, e.g., $[p_1 + 2p_2 + p_3]$. This is particularly useful for markings having only one token, e.g., $[p]$ is the marking with just a token in place p.

To compare markings, we define a partial ordering. For any two markings M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

The number of tokens may change during the execution of the net. Transitions are the active components in a net: they change the marking of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* a token from each input place p of t and *produces* a token for each output place p of t.

A *system* is a tuple $N = (PN, M_0)$, where PN is a Petri net and M_0 is an *initial* marking. Although there is a clear distinction between a marked system and an unmarked Petri net, in text we will sometimes also use the term "Petri net" to refer to a system, i.e., the network structure and its marking.

A *labelled* Petri net is a tuple (P, T, F, L) where (P, T, F) is a Petri net and L is a mapping that associates to each transition t a label $L(t)$ taken from some given set of actions $\mathcal{N}$. A *labelled* system is a tuple (P, T, F, L, M_0) where (P, T, F, L) is a labelled Petri net and M_0 an initial marking.

Note that labelled nets can be mapped onto unlabelled nets by removing the labels and that unlabelled nets can be mapped onto labelled nets by adding a dummy label. Therefore, we will use them interchangeably.

Given a labelled system $PN = (P, T, F, L, M_0)$ and a marking M_1, we have the following notations:

- $M_1 \xrightarrow{t}_{PN} M_2$: transition t is enabled in marking M_1 and firing t in M_1 results in marking M_2
- $M_1 \xrightarrow{a}_{PN} M_2$: a transition t with $L(t) = a$ is enabled in marking M_1 and firing t in M_1 results in marking M_2
- $M_1 \longrightarrow_{PN} M_2$: there is a transition t such that $M_1 \xrightarrow{t}_{PN} M_2$
- $M_1 \xrightarrow{\sigma}_{PN} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1} \in T^*$ leads from marking M_1 to marking M_n, i.e., $M_1 \xrightarrow{t_1}_{PN} M_2 \xrightarrow{t_2}_{PN} \ldots \xrightarrow{t_{n-1}}_{PN} M_n$

A marking M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*}_{PN} M_2$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1 \xrightarrow{\sigma}_{PN} M_n$. The subscript $_{PN}$ is omitted if it is clear which Petri net is considered. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*}_{PN} M_1$.

A marking M is a *reachable marking* of a (labelled) system (PN, M_0) iff $M_0 \xrightarrow{*} M$.

A marking M_h is *home marking* of (PN, M_0) iff for every reachable marking M, $M \xrightarrow{*} M_h$

(PN, M_0) is *safe* iff $M(p) \leq 1$ for every place p and every reachable marking M.

(PN, M_0) is *bounded* iff the set of reachable markings is finite.

A (labelled) Petri net is *free-choice* iff $\forall_{t \in T, p \in P} [(p,t) \in F \Rightarrow \bullet t \times p\bullet \subseteq F]$.

References

[AAH98] N. R. Adam, V. Atluri, and W. Huang. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management*, 10(2):131–158, March/April 1998.

[Aal98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[Aal99] W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[ABHK00] W.M.P. van der Aalst, A.P. Barros, A.H.M. ter Hofstede, and B. Kiepuszewski. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, *Fifth IFCIS International Conference on Cooperative Information Systems (CoopIS'2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29, Eilat, Israel, September 2000. Springer-Verlag.

[AH02] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.

[AHKB00] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Technical Report WP 47, BETA Research Institute, Eindhoven University of Technology, Eindhoven, The Netherlands, August 2000.

[AHKB02] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02 (to appear in Distributed and Parallel Databases), Queensland University of Technology, Brisbane, 2002. http://www.tm.tue.nl/it/research/patterns.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, United Kingdom, 1990.

[Cas98] F. Casati. *Semantics and Formal methods for the Design of Workflows and their Exceptions*. PhD thesis, Politecnico di Milano, Milano, Italy, 1998.

[CPP98] F. Casati, F.S. Ceri B. Pernici, and G. Pozzi. Conceptual Modeling of Workflows. In M.P. Papazoglou, editor, *Proceedings of the 14th International Object-Oriented and Entity-Relationship Modeling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer-Verlag, Berlin, 1998.

[DE95] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, United Kingdom, 1995.

[DR01] J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer-Verlag, Berlin, 2001.

[Ell79] C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.

[Fis01] L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition.* Future Strategies, Lighthouse Point, Florida, 2001.

[Gen87] H.J. Genrich. Predicate/Transition Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer-Verlag, Berlin, Germany, 1987.

[Gla90] R.J. van Glabbeek. The linear time-branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR'90. Theories of Concurrency: Unification and Extension*, pages 278–297, Berlin, Germany, 1990. Springer-Verlag.

[Gla93] R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, Proceedings *CONCUR'93,* 4^{th} International Conference on Concurrency Theory, Hildesheim, Germany, August 1993, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[Gla94] R.J. van Glabbeek. What is Branching Time Semantics and Why to Use It? In M. Nielsen, editor, *The Concurrency Column*, pages 190–198. *Bulletin of the EATCS* 53, 1994. Also available as Report STAN-CS-93-1486, Stanford University, 1993, and by ftp at `ftp://Boole.stanford.edu/pub/branching.ps.gz`.

[GT84] H.J. Genrich and P. S. Thiagarajan. A Theory of Bipolar Synchronization Schemes. *Theoretical Computer Science*, 30(3):241–318, 1984.

[GV87] R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency (extended abstract). In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE, Parallel Architectures and Languages Europe, Vol. II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242, Eindhoven, The Netherlands, June 1987. Springer Verlag.

[HO99] A.H.M. ter Hofstede and M.E. Orlowska. On the Complexity of Some Verification Problems in Process Control Specifications. *Computer Journal*, 42(5):349–359, 1999.

[HOR98] A.H.M. ter Hofstede, M.E. Orlowska, and J. Rajapakse. Verification Problems in Conceptual Workflow Specifications. *Data & Knowledge Engineering*, 24(3):239–256, January 1998.

[JB96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation.* International Thomson Computer Press, London, UK, 1996.

[Jen87] K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299, Berlin, Germany, 1987. Springer-Verlag.

[Kie02] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows (submitted).* PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002. Available via http://www.tm.tue.nl/it/research/patterns.

[KNS92] G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.

[LR99] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1999.

[LSW98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer-Verlag, Berlin, 1998.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: The Pi Calcalus*, volume 92. Cambridge University Press, Cambridge, UK, 1999.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77:541–580, 1989.

[PRS92] L. Pomello, G. Rozenberg, and C. Simone. A Survey of Equivalence Notions of Net Based Systems. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer, 1992.

[Rit99] P. Rittgen. Modified EPCs and their Formal Semantics. Technical report 99/19, University of Koblenz-Landau, Koblenz, Germany, 1999.

[RR98] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[Rum97] F. Rump. Erreichbarkeitsgraphbasierte Analyse ereignisgesteuerter Prozessketten. Technischer Bericht, Institut OFFIS, 04/97 (in German), University of Oldenburg, Oldenburg, 1997.

[SH96] P. Straub and C. Hurtado. Business Process Behavior is (Almost) Free-Choice. In *Proceedings of the IMACS-IEEE Multiconference on Computational Engineering in Systems Applications (CESA'96)*, Lille, France, July 1996.

[Smi96] E. Smith. On the border of causality: contact and confusion. *Theoretical Computer Science*, 153:245–270, 1996.

[Ver00] Verve. *Verve Component Workflow Engine Concepts*. Verve, Inc., San Francisco, CA, USA, 2000.

[Wor99] Workflow Management Coalition. Terminology & Glossary. Document Number WFMC-TC-1011, Document Status - Issue 3.0, February 1999. www.wfmc.org.

[WPH02] Workflow Patterns Home Page, 2002. http://www.tm.tue.nl/it/research/patterns.