# Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management

Wil M.P. van der Aalst

Department of Technology Management
Eindhoven University of Technology
P.O.Box 513, NL-5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

**Abstract.** Over the last decade there has been a shift from "data-aware" information systems to "process-aware" information systems. To support business processes an enterprise information system needs to be aware of these processes and their organizational context. Business Process Management (BPM) includes methods, techniques, and tools to support the design, enactment, management, and analysis of such operational business processes. BPM can be considered as an extension of classical Workflow Management (WFM) systems and approaches. This tutorial introduces models, systems, and standards for the design, analysis, and enactment of workflow processes. Petri nets are used for the modeling and analysis of workflows. Using Petri nets as a formal basis, contemporary systems, languages, and standards for BPM and WFM are discussed. Although it is clear that Petri nets can serve as a solid foundation for BPM/WFM technology, in reality systems, languages, and standards are developed in an ad-hoc fashion. To illustrate this XPDL, the "Lingua Franca" proposed by the Workflow Management Coalition (WfMC), is analyzed using a set of 20 basic workflow patterns. This analysis exposes some of the typical semantic problems restricting the application of BPM/WFM technology.

**Keywords:** Business process management, Workflow management, Workflow management systems, Workflow patterns, XML Process Definition Language (XPDL), Workflow verification.

## 1 Introduction

This section provides some context for the topics addressed in this tutorial. First, we identify some trends and put them in a historical perspective. Then, we focus on the BPM life-cycle and discuss the basic functionality of a WFM system. Finally, we outline the remainder of this tutorial.

### 1.1 Historical perspective

To show the relevance of Business Process Management (BPM) systems, it is interesting to put them in a historical perspective. Consider Figure 1, which shows some of the

ongoing trends in information systems. This figure shows that today's information systems consist of a number of layers. The center is formed by the operating system, i.e., the software that makes the hardware work. The second layer consists of generic applications that can be used in a wide range of enterprises. Moreover, these applications are typically used within multiple departments within the same enterprise. Examples of such generic applications are a database management system, a text editor, and a spreadsheet program. The third layer consists of domain specific applications. These applications are only used within specific types of enterprises and departments. Examples are decision support systems for vehicle routing, call center software, and human resource management software. The fourth layer consists of tailor-made applications. These applications are developed for specific organizations.

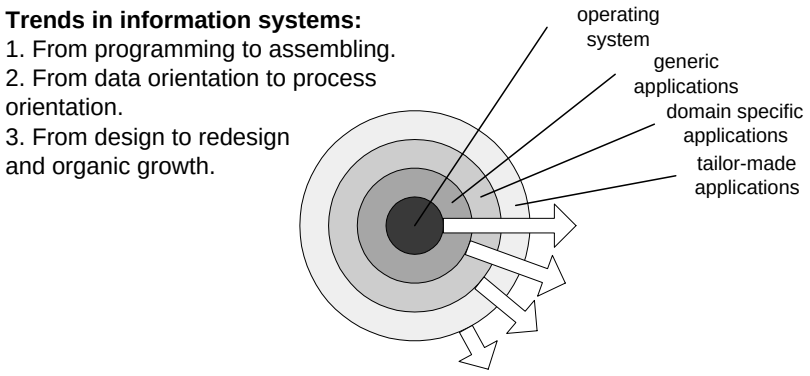


**Fig. 1.** Trends relevant for business process management.

In the sixties the second and third layer were missing. Information systems were built on top of a small operating system with limited functionality. Since no generic nor domain specific software was available, these systems mainly consisted of tailor-made applications. Since then, the second and third layer have developed and the ongoing trend is that the four circles are increasing in size, i.e., they are moving to the outside while absorbing new functionality. Today's operating systems offer much more functionality. Database management systems that reside in the second layer offer functionality which used to be in tailor-made applications. As a result of this trend, the emphasis shifted from programming to assembling of complex software systems. The challenge no longer is the coding of individual modules but orchestrating and gluing together pieces of software from each of the four layers.

Another trend is the shift from data to processes. The seventies and eighties were dominated by data-driven approaches. The focus of information technology was on storing and retrieving information and as a result data modeling was the starting point for building an information system. The modeling of business processes was often neglected and processes had to adapt to information technology. Management trends such

as business process reengineering illustrate the increased emphasis on processes. As a result, system engineers are resorting to a more process driven approach.

The last trend we would like to mention is the shift from carefully planned designs to redesign and organic growth. Due to the omnipresence of the Internet and its standards, information systems change on-the-fly. As a result, fewer systems are built from scratch. In many cases existing applications are partly used in the new system. Although component-based software development still has it problems, the goal is clear and it is easy to see that software development has become more dynamic.

The trends shown in Figure 1 provide a historical context for BPM. BPM systems are either separate applications residing in the second layer or are integrated components in the domain specific applications, i.e., the third layer. Notable examples of BPM systems residing in the second layer are WorkFlow Management (WFM) systems [12, 38, 48, 55, 57, 58, 61] such as Staffware, MQSeries, and COSA, and case handling systems such as FLOWer. Note that leading Enterprise Resource Planning (ERP) systems populating the third layer also offer a WFM module. The workflow engines of SAP, Baan, PeopleSoft, Oracle, and JD Edwards can be considered as integrated BPM systems. The idea to isolate the management of business processes in a separate component is consistent with the three trends identified. BPM systems can be used to avoid hard-coding the work processes into tailor-made applications and thus support the shift from programming to assembling. Moreover, process orientation, redesign, and organic growth are supported. For example, today's WFM systems can be used to integrate existing applications and support process change by merely changing the workflow diagram. Given these observations, the practical relevance of BPM is evident. Although BPM functionality is omnipresent and often hidden in larger enterprise information systems, for clarity we will often restrict the discussion to clear cut "process-aware" information systems such as WFM systems (cf. Section 1.3).

To put the topic of this tutorial in a historical perspective it is worthwhile to consider the early work on office information systems. In the seventies, people like Skip Ellis [32], Anatol Holt [45], and Michael Zisman [78] already worked on so-called office information systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the eighties. In the nineties, there again was a huge interest in these systems. The number of WFM systems developed in the past decade and the many papers on workflow technology illustrate the revival of office information systems. Today WFM systems are readily available [12, 38, 48, 55, 57, 58, 61]. However, their application is still limited to specific industries such as banking and insurance. As was indicated by Skip Ellis it is important to learn from these ups and downs [33]. The failures in the eighties can be explained by both technical and conceptual problems. In the eighties, networks were slow or not present at all, there were no suitable graphical interfaces, and proper development software was missing. However, there were also more fundamental problems: a unified way of modeling processes was missing and the systems were too rigid to be

used by people in the workplace. Most of the technical problems have been resolved by now. However, the more conceptual problems remain. Good standards for business process modeling are still missing and even today's WFM systems enforce unnecessary constrains on the process logic (e.g., processes are made more sequential).

### 1.2 BPM life-cycle

As indicated before, *Business Process Management (BPM) includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes*. It can be considered as an extension of classical Workflow Management (WFM) systems and approaches. Before discussing the differences between WFM and BPM, let us consider the *BPM life-cycle*.

The BPM life-cycle has four phases:

- *Process design*
  Any BPM effort requires the modeling of an existing ("as-is") or desired ("to-be") process, i.e., a *process design*. During this phase process models including various perspectives (control-flow, data-flow, organizational, sociotechnical, and operational aspects) are constructed. The only way to create a "process-aware" enterprise information system is to add knowledge about the operational processes at hand.
- *System configuration*
  Based on a process design, the process-aware enterprise information system is realized. In the traditional setting the realization would require a time-consuming and complex software development process. Using software from the second and third layer shown in Figure 1, the traditional software development process is replaced by a configuration or assembly process. Therefore, we use the term *system configuration* for the phase in-between process design and enactment.
- *Process enactment*
  The *process enactment* phase is the phase where the process-aware enterprise information system realized in the system configuration phase is actually used.
- *Diagnosis*
  Process-aware enterprise information system have to change over time to improve performance, exploit new technologies, support new processes, and adapt to an ever changing environment. Therefore, the *diagnosis* phase is linking the process enactment phase to the a new design phase.

Like in software life-cycle models, the four phases are overlapping (cf. Waterfall model) and the whole process is iterative (cf. Spiral model).

As is illustrated in Figure 2, the BPM life-cycle can be used to identify different levels of maturity when it comes to developing process-aware enterprise information systems. In the early nineties and before, most information systems only automated individual activities and where unaware of the underlying process. For the systems that were process-aware, the process logic was hard-coded in the system and not supported in a generic manner. Despite the early work on office automation, the first commercial WFM systems became only practically relevant around 1993 (see Figure 2(a)). The focus of these systems was on "getting the system to work" and support for enactment and

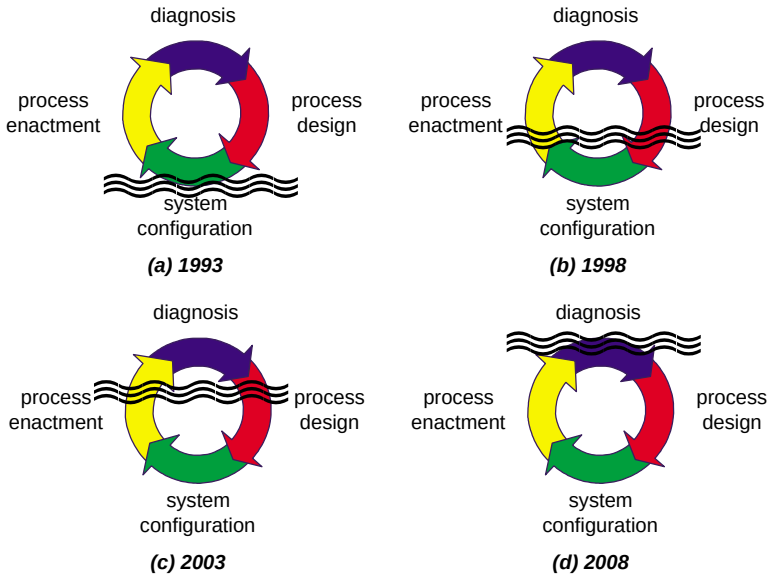


**Fig. 2.** The BPM life-cycle is used to indicate the maturity of BPM technology over time.

design was limited. In the mid-nineties this situation changed and by 1998 many WFM systems had become readily available (see Figure 2(b)). In these systems there was basic support for enactment and design. In the last five years these systems have been further extended allowing for more support during the design and enactment phases (see Figure 2(c)). For example, a case-handling system like FLOWer [22] allows for much more flexibility during the enactment phase than the traditional WFM systems. Today‘s systems provide hardly any support for the diagnosis phase. Although most BPM software logs all kinds of events (e.g., WFM systems like Staffware log the completion of activities and ERP systems like SAP log transactions), this information is not used to identify problems or opportunities for improvement. In the next five years this situation will probably change when process mining [17, 19] techniques become readily available (see Figure 2(d)).

The BPM life-cycle shown in Figure 2 can also be used to define the difference between WFM and BPM. WFM focusses on the lower half of the BPM life-cycle (i.e., “getting the system to work”) while BPM also includes to upper half of the life-cycle. Therefore, BPM also focusses on diagnosis, flexibility, human-centric processes, goal-driven process design, etc. Gartner expects that *Business Process Analysis* (BPA), i.e., software to support the diagnosis phase, will become increasingly important [39]. It is expected that the BPA market will continue to grow. Note that BPA covers aspects neglected by traditional WFM products (e.g., diagnosis, simulation, etc.). *Business Activity Monitoring* (BAM) is one of the emerging areas in BPA. The goal of BAM tools is to use data logged by the information system to diagnose the operational processes. An example is the ARIS Process Performance Manager (PPM) of IDS Scheer [47].

ARIS PPM extracts information from audit trails (i.e., information logged during the execution of cases) and displays this information in a graphical way (e.g., flow times, bottlenecks, utilization, etc.). BAM also includes process mining, i.e., extracting process models from logs [17]. BAM creates a number of scientific and practical challenges (e.g., which processes can be discovered and how much data is needed to provide useful information).

### 1.3 Workflow management (systems)

The focus of this tutorial will be on WFM rather than BPM. The reason is that WFM serves as a basis for BPM and in contrast to BPM it is a mature area with well-defined concepts and widely used software products.

The Workflow Management Coalition (WfMC) defines workflow as: "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [55]. A Workflow Management System (WFMS) is defined as: "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." [55]. Note that both definitions emphasize the focus on enactment, i.e., the use of software to support the execution of operational processes.

When considering these definitions in more detail it is evident that WFM is highly relevant for any organization. However, at the same time few organizations use a "real" WFM system. To explain this we identify four categories of WFM support:

- *Pure WFM systems*
  At this point in time many WFM systems are available and used in practise. Examples of systems include Staffware Process Suite, FileNET BPM Suite, i-Flow, FLOWer, WebSphere MQ Workflow (formerly known as MQSeries Workflow), TIBCO InConcert, etc.
- *WFM components embedded in other systems*
  Many software packages embed a generic workflow component whose functionality is comparable to the pure WFM systems. For example, most ERP systems provide a workflow component. SAP WebFlow is the workflow component of SAP offering all the functionality typically present in traditional stand-alone WFM products.
- *Custom-made WFM solutions*
  Many organizations, e.g., banks and insurance companies, have chosen not to use a commercially available WFM solution but build an organization-specific solution. These solutions typically only support a subset of the functionality offered by the first two categories. Nevertheless, these systems support the definition and execution of different workflows.
- *Hard-coded WFM solutions*
  The last category refers to the situation were the processes are hard-coded in the applications, i.e., there is no generic workflow support but applications are coupled in such a way that a specific process is supported. The only way to change a process

is to change the applications themselves, i.e., unlike the first three categories there is no component that is process-aware. Note that in these hard-coded system an explicit orchestration layer is missing.

At this point in time the majority of business processes are still supported by solutions residing in the third and fourth category. However, the percentage of processes supported by the first two categories is increasing. Moreover, software developers building solutions for the third and fourth category are using the concepts and insights provided by the first two categories. In this context it is interesting to refer to recent developments in the *web services* domain [68]. The functionality of web service composition languages (also referred to as "web service orchestration") like BPEL4WS, BPML, WSCI, WSWSFL, XLANG, etc. is very similar to traditional workflow languages [6, 77].

### 1.4 Outline and intended audience

The goal of this tutorial is to introduce the reader to the theoretical foundations of BPM/WFM using a Petri-net based approach. However, at the same time contemporary systems and languages are presented to provide a balanced view on the application domain.

Section 2 shows the application of Petri nets to workflow modeling. For this purpose, the class of *WorkFlow nets* (WF-nets) is introduced, but also some "syntactical sugaring" is given to facilitate the design of workflows. Section 3 discusses the analysis of workflow models expressed in terms of Petri nets. The focus will be on the verification of WF-nets using classical analysis techniques. Section 4 discusses the typical architecture of a WFM system and discusses contemporary systems. The goal of this section is to show that the step from design to enactment, i.e., the configuration phase (cf. Figure 2), is far from trivial. In Section 5, 20 workflow patterns are used to evaluate the XML Process Definition Language (XPDL), the standard proposed by the Workflow Management Coalition (WfMC). This evaluation illustrates the typical problems workflow designers and implementers are faced with when applying contemporary languages and standards. Section 6 provides an overview of related work. Clearly, only a small subset of the many books and papers on BPM/WFM can be presented, but pointers are given to find relevant material. Finally, Section 7 concludes the tutorial by discussing the role of Petri nets in the BPM/WFM domain.

Note that parts of this tutorial are based on earlier work (cf. [2–6, 12, 15]). For more material the interested reader is referred to [12] and two WWW-sites: one presenting course material (slides, animations, etc.) `http://www.workflowcourse.com` and one on workflow patterns `http://www.workflowpatterns.com`.

This tutorial is intended for people having a basic understanding of Petri nets and interested in the application of Petri nets to problems in the BPM/WFM domain. Sections 2 and 3 are focusing more on the Petri-net side of things while sections 4 and 5 are focusing more on the application domain.

## 2 Workflow modeling

In this section, we show how to model workflows in terms of Petri nets. First, we introduce the basic workflow concepts and discuss the various perspectives. Then, we define some basic Petri net notation followed by an introduction to a subclass of Petri nets tailored towards workflow modeling. We conclude this section with an exercise.

### 2.1 Workflow concepts and perspectives

Workflow processes are *case-driven*, i.e., tasks are executed for specific cases. Approving loans, processing insurance claims, billing, processing tax declarations, handling traffic violations and mortgaging, are typical case-driven processes which are often supported by a WFM system. These case-driven processes, also called *workflows*, are marked by three dimensions: (1) the control-flow dimension, (2) the resource dimension, and (3) the case dimension (see Figure 3). The control-flow dimension is concerned with the partial ordering of tasks, i.e., the workflow *process*. The tasks which need to be executed are identified and the routing of cases along these tasks is determined. Conditional, sequential, parallel and iterative routing are typical structures specified in the control-flow dimension. Tasks are executed by resources. Resources are human (e.g., employee) and/or non-human (e.g., device, software, hardware). In the resource dimension these resources are classified by identifying roles (resource classes based on functional characteristics) and organizational units (groups, teams or departments). Both the control-flow dimension and the resource dimension are generic, i.e., they are not tailored towards a specific case. The third dimension of a workflow is concerned with individual cases which are executed according to the process definition (first dimension) by the proper resources (second dimension).

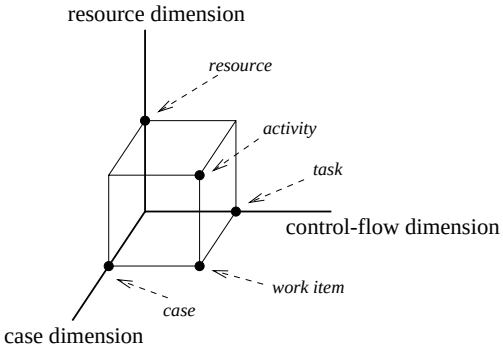


**Fig. 3.** The three dimensions of workflow.

The primary task of a WFM system is to enact case-driven business processes by joining several perspectives. The following perspectives are relevant for workflow modeling and workflow execution: (1) *control flow* (or process) perspective, (2) *resource* (or

organization) perspective, (3) *data* (or information) perspective, (4) *task* (or function) perspective, (5) *operation* (or application) perspective. These perspectives are similar to the perspectives given in [48] and the control flow and resource perspectives correspond to the first two dimensions shown in Figure 3. The third dimension reflects the fact that workflows are case-driven and does not correspond to one of the five perspectives.

In the control-flow perspective, *workflow process definitions* (workflow schemas) are defined to specify which *tasks* need to be executed and in what order (i.e., the routing or control flow). A task is an atomic piece of work. Workflow process definitions are instantiated for specific *cases* (i.e., workflow instances). Since a case is an instantiation of a process definition, it corresponds to the execution of concrete work according to the specified routing. In the *resource* perspective, the organizational structure and its population are specified. The organizational structure describes relations between roles (resource classes based on functional aspects) and groups (resource classes based on organizational aspects). Thus clarifying organizational issues such as responsibility, availability, and authorization. Resources, ranging from humans to devices, form the organizational population and are allocated to roles and groups. The data perspective deals with *control* and *production data*. Control data are data introduced solely for WFM purposes, e.g., variables introduced for routing purposes. Production data are information objects (e.g., documents, forms, and tables) whose existence does not depend on WFM. The task perspective describes the elementary operations performed by resources while executing a task for a specific case. In the operational perspective the elementary actions are described. These actions are often executed using applications ranging from a text editor to custom build applications to perform complex calculations. Typically, these applications create, read, or modify control and production data in the information perspective.

The focus of this tutorial will be on the control-flow perspective. Clearly, this is the most dominant perspective. Moreover, Petri nets can contribute most to this perspective.

### 2.2 Petri nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.[1]

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

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

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

[1] Note that states are represented by weighted sums and note the definition of (elementary) (conflict-free) paths.

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. Note that we do not consider multiple arcs from one node to another. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state 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 state as follows: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states $M_1$ 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 Petri net: they change the state 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* one token from each input place $p$ of $t$ and *produces* one token for each output place $p$ of $t$.

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

- $M_1 \stackrel{t}{\rightarrow} M_2$: transition $t$ is enabled in state $M_1$ and firing $t$ in $M_1$ results in state $M_2$
- $M_1 \rightarrow M_2$: there is a transition $t$ such that $M_1 \stackrel{t}{\rightarrow} M_2$
- $M_1 \stackrel{\sigma}{\rightarrow} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state $M_1$ to state $M_n$ via a (possibly empty) set of intermediate states $M_2, \ldots M_{n-1}$, i.e., $M_1 \stackrel{t_1}{\rightarrow} M_2 \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} M_n$

A state $M_n$ is called *reachable* from $M_1$ (notation $M_1 \stackrel{*}{\rightarrow} M_n$) iff there is a firing sequence $\sigma$ such that $M_1 \stackrel{\sigma}{\rightarrow} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \stackrel{*}{\rightarrow} M_1$.

We use $(PN, M)$ to denote a Petri net $PN$ with an initial state $M$. A state $M'$ is a *reachable state* of $(PN, M)$ iff $M \stackrel{*}{\rightarrow} M'$.

Let us define some standard properties for Petri nets. First, we define properties related to the dynamics of a Petri net, then we give some structural properties.

**Definition 2 (Live).** *A Petri net $(PN, M)$ is live iff, for every reachable state $M'$ and every transition $t$ there is a state $M''$ reachable from $M'$ which enables $t$.*

A Petri net is *structurally live* if there exists an initial state such that the net is live.

**Definition 3 (Bounded, safe).** *A Petri net $(PN, M)$ is bounded iff for each place $p$ there is a natural number $n$ such that for every reachable state the number of tokens in $p$ is less than $n$. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

A Petri net is *structurally bounded* if the net is bounded for any initially state.

**Definition 4 (Well-formed).** *A Petri net $PN$ is well-formed iff there is a state $M$ such that $(PN, M)$ is live and bounded.*

Paths connect nodes by a sequence of arcs.

**Definition 5 (Path, Elementary, Conflict-free).** *Let $PN$ be a Petri net. A path $C$ from a node $n_1$ to a node $n_k$ is a sequence $\langle n_1, n_2, \ldots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$. $C$ is elementary iff, for any two nodes $n_i$ and $n_j$ on C, $i \neq j \Rightarrow n_i \neq n_j$. $C$ is conflict-free iff, for any place $n_j$ on $C$ and any transition $n_i$ on $C$, $j \neq i-1 \Rightarrow n_j \notin {\bullet}n_i$.*

For convenience, we introduce the alphabet operator $\alpha$ on paths. If $C = \langle n_1, n_2, \ldots, n_k \rangle$, then $\alpha(C) = \{n_1, n_2, \ldots, n_k\}$.

**Definition 6 (Strongly connected).** *A Petri net is strongly connected iff, for every pair of nodes (i.e., places and transitions) $x$ and $y$, there is a path leading from $x$ to $y$.*

**Definition 7 (Free-choice).** *A Petri net is a free-choice Petri net iff, for every two transitions $t_1$ and $t_2$, ${\bullet}t_1 \cap {\bullet}t_2 \neq \emptyset$ implies ${\bullet}t_1 = {\bullet}t_2$.*

**Definition 8 (State machine).** *A Petri net is state machine iff each transition has exactly one input and one output place.*

**Definition 9 (S-component).** *A subnet $PN_s = (P_s, T_s, F_s)$ is called an S-component of a Petri net $PN = (P, T, F)$ if $P_s \subseteq P$, $T_s \subseteq T$, $F_s \subseteq F$, $PN_s$ is strongly connected, $PN_s$ is a state machine, and for every $q \in P_s$ and $t \in T$: $(q, t) \in F \Rightarrow (q, t) \in F_s$ and $(t, q) \in F \Rightarrow (t, q) \in F_s$.*

**Definition 10 (S-coverable).** *A Petri net is S-coverable iff for any node there exist an S-component which contains this node.*

See [30, 63] for a more elaborate introduction to these standard notions.

### 2.3 WF-nets

In Figure 3 we indicated that a workflow has (at least) three dimensions. The control-flow dimension is the most prominent one, because the core of any workflow system is formed by the processes it supports. In the control-flow dimension building blocks such as the AND-split, AND-join, OR-split, and OR-join are used to model sequential, conditional, parallel and iterative routing [55]. Clearly, a Petri net can be used to specify the routing of cases. *Tasks* are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-condition for tasks. 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. Moreover, in [2] it is shown that the Petri net approach also allows for useful routing constructs absent in many WFM systems.

A Petri net which models the control-flow dimension of a workflow, is called a *WorkFlow net* (WF-net). It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation.

**Definition 11 (WF-net).** *A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if:*

*(i) There is one source place $i \in P$ such that $\bullet i = \emptyset$.*
*(ii) There is one sink place $o \in P$ such that $o\bullet = \emptyset$.*
*(iii) Every node $x \in P \cup T$ is on a path from $i$ to $o$.*

A WF-net has one input place ($i$) and one output place ($o$) because any case handled by the procedure represented by the WF-net is created when it enters the WFM system and is deleted once it is completely handled by the system, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 11 has been added to avoid "dangling tasks and/or conditions", i.e., tasks and conditions which do not contribute to the processing of cases.

Given the definition of a WF-net it is easy derive the following properties.

**Proposition 1 (Properties of WF-nets).** *Let $PN = (P, T, F)$ be Petri net.*

- *If $PN$ is WF-net with source place $i$, then for any place $p \in P$: $\bullet p \neq \emptyset$ or $p = i$, i.e., $i$ is the only source place.*
- *If $PN$ is WF-net with sink place $o$, then for any place $p \in P$: $p\bullet \neq \emptyset$ or $p = o$, i.e., $o$ is the only sink place.*
- *If $PN$ is a WF-net and we add a transition $t^*$ to $PN$ which connects sink place $o$ with source place $i$ (i.e., $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.*
- *If $PN$ has a source place $i$ and a sink place $o$ and adding a transition $t^*$ which connects sink place $o$ with source place $i$ yields a strongly connected net, then every node $x \in P \cup T$ is on a path from $i$ to $o$ in $PN$ and $PN$ is a WF-net.*

Figure 4 shows an example of an order handling process modeled in terms of a WF-net. As indicated before cases are represented by tokens and in Figure 4 the token in *start* corresponds to an order. Task *register* is represented by a transition bearing the same name. From a routing point of view it acts as a so-called AND-split (two outgoing arcs) and is enabled in the state shown. If a person executes this task, the token is removed from place *start* and two tokens are produced: one for *c0* and one for *c2*. Then, in parallel, two tasks are enabled: *check_availability* and *send_bill*. Depending on the eagerness of the workers executing these two tasks either *check_availability* or *send_bill* is executed first. Suppose *check_availability* is executed first. Based on the outcome of this task a choice is made. This is reflected by the fact that three arcs are leaving *c1*. If the ordered goods are available, they can be shipped, i.e., firing *in_stock* enables task *ship_goods*. If they are not available, either a replenishment order is issued or not. Firing *out_of_stock_repl* enables task *replenish*. Firing *out_of_stock_no_repl* skips task *replenish*. Note that *check_availability*, place *c1* and the three transitions *in_stock*, *out_of_stock_repl*, and *out_of_stock_no_repl* together form a so-called OR-split: As a result of this construct one token is produced for either *c3*, *c4*, or *c5*. Suppose that not all ordered goods are available, but the appropriate replenishment orders were already issued. A token is produced for *c3* and task *update* becomes enabled. Suppose that at this point in time task *send_bill* is executed, resulting in the state with a token in *c3* and *c6*. The token in *c6* is input for two tasks. However, only one of these tasks can be

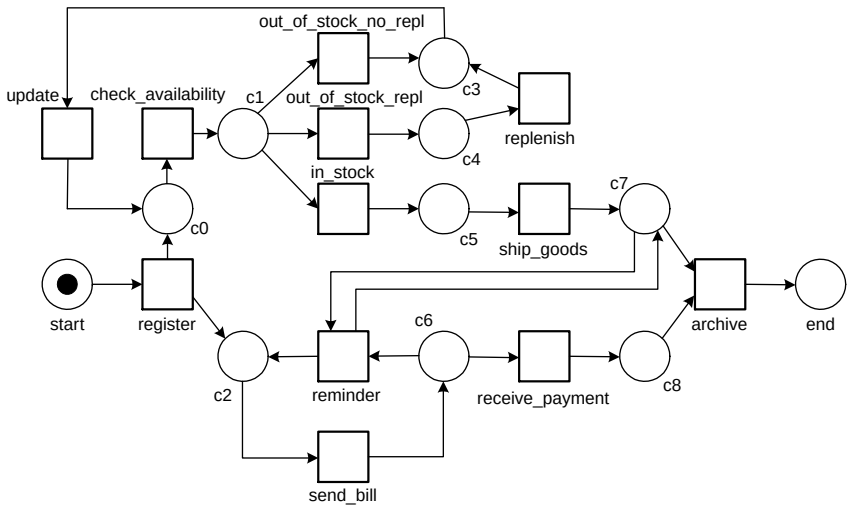


**Fig. 4.** WF-net.

executed and in this state only *receive_payment* is enabled. Task *receive_payment* can be executed the moment the payment is received. Task *reminder* is an AND-join/AND-split and is blocked until the bill is sent and the goods have been shipped. However, it is only possible to send a reminder if the goods have been actually shipped. Assume that in the state with a token in *c3* and *c6* task *update* is executed. This task does not require human involvement and is triggered by a message of the warehouse indicating that relevant goods have arrived. Again *check_availability* is enabled. Suppose that this task is executed and the result is positive, i.e., the path via *in_stock* is taken. In the resulting state *ship_goods* can be executed. Now there is a token in *c6* and *c7* thus enabling task *reminder*. Executing task *reminder* enables the task *send_bill* for the second time. A new copy of the bill is sent with the appropriate text. It is possible to send several reminders by alternating *reminder* and *send_bill*. However, let us assume that after the first loop the customer pays resulting in a state with a token in *c7* and *c8*. In this state, the AND-join *archive* is enabled and executing this task results in the final state with a token in *end*.

Figure 4 shows some of the limitations of WF-nets. First of all, the construct involving *check_availability*, place *c1* and the three transitions *in_stock*, *out_of_stock_repl*, and *out_of_stock_no_repl* is rather complex for a simple concept as a choice out of three alternatives. Second, the diagram does not show *why* things are happening. The text suggests that some of the tasks are executed by people while others are triggered by external entities or temporal conditions. Unfortunately, this information is missing in Figure 4. Finally, the WF-net does not show the other perspectives. The first two problems can be solved using some “syntactical sugaring” (cf. Figure 5). The third problem will not be addressed in this tutorial. Here we abstract from the other perspectives. The interested reader is referred to [12] for modeling the resource perspective.

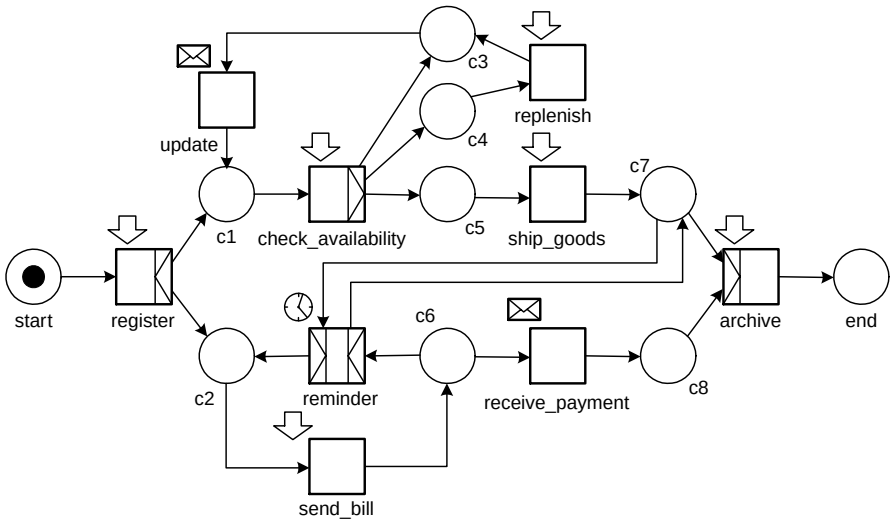


**Fig. 5.** WF-net extended with some "syntactical sugaring" to denote AND/OR-splits/joins and triggers.

Figure 5 shows the same process as the one depicted in Figure 4. However, every task can be an AND/OR-join - AND/OR-split. The semantics of a transition is AND-join - AND-split. Choices can be modeled using places with multiple outgoing arcs. However, the intuition of a task resulting in a choice is better reflected by the notation used in Figure 5: the construct involving *check_availability*, place *c1* and the three transitions *in_stock*, *out_of_stock_repl*, and *out_of_stock_no_repl* is replaced by a single task *check_availability* using the notation for an OR-split. Note that any WF-net with OR-splits can be automatically translated into standard WF-net (i.e., a classical Petri net). Figure 5 also shows three triggers symbols: (1) an arrow denoting a user trigger, (2) an envelope denoting an external trigger, and (3) a clock denoting a time trigger. These three triggers symbols denote that the corresponding tasks need a trigger to be executed, e.g., the tasks bearing an arrow symbol require a user to perform the corresponding activity. Task *receive payment* can only be executed after the payment trigger arrives. Task *reminder* can only be executed after a specified period. Although triggers are extremely important, we will not formalize the concept. For the reader interested in the topic we refer to [28, 34] for a discussion on the reactive nature of WFM systems.

The very simple WF-net shown in Figure 5 shows some of the routing constructs relevant for business process modeling. Sequential, parallel, conditional, and iterative routing are present in this model. There are also more advanced constructs such as the choice between *receive_payment* and *reminder*. This is a so-called *deferred choice* (also referred to as implicit choice) since it is not resolved by the system but by the environment of the system. The moment the bill is sent, it is undetermined whether *receive_payment* or *reminder* will be the next step in the process. Another advanced construct is the fact that task *reminder* is blocked until the goods have been shipped. The

latter construct is a so-called *milestone*. The reason that we point out both constructs is that many systems have problems supporting these rather fundamental process patterns. In Section 5.1 we will discuss these patterns in more detail.

### 2.4 Exercise: Modeling a complaints handling process in terms of a WF-net

To conclude this section, we give a small exercise. Model the complaints handling workflow of a travel agency in terms of a WF-net, i.e., construct a diagram similar to Figure 5.

Each year the travel agency has to process many customer complaints. There is a special department for the processing of complaints (department C). There is also an internal department called logistics (department L) which takes care of the registration of incoming complaints and the archiving of processed complaints. The following procedure is used to handle these complaints.

An employee of department L first registers every incoming complaint. After registration a form is sent to the customer with questions about the nature of the complaint. This is done by an employee of department C. There are two possibilities: the customer returns the form within two weeks or not. If the form is returned, it is processed automatically resulting in a report which can be used for the actual processing of the complaint. If the form is not returned on time, a time-out occurs resulting in an empty report. Note that this does not necessarily mean that the complaint is discarded. After registration, i.e., in parallel with the form handling, the preparation for the actual processing is started.

First, the complaint is evaluated by a complaint manager of department C. Evaluation shows that either further processing is needed or not. Note that this decision does not depend on the form handling. If no further processing is required and the form is handled, the complaint is archived. If further processing is required, an employee of the complaints department executes the task "process complaint" (this is the actual processing where certain actions are proposed if needed). For the actual processing of the complaint, the report resulting from the form handling is used. Note that the report can be empty. The result of task process complaint is checked by a complaint manager. If the result is not OK, task process complaint is executed again. This is repeated until the result is acceptable. If the result is accepted, an employee of the department C executes the proposed actions. After this the processed complaint is archived by an employee of department L.

Give the WF-net, i.e., model the workflow by making a process definition in terms of a Petri net. For the solution to this exercise we refer to [12] or the corresponding WWW site with course material: `http://www.workflowcourse.com`.

## 3 Workflow analysis

One of the advantages of using Petri nets for workflow modeling is the availability of many Petri-net-based analysis techniques. In this section, we focus on the verification of WF-nets. The correctness criterion used is the so-called *soundness property*. We will show how this property can be checked and discuss a verification tool specifically designed for workflow analysis.

### 3.1 Verification, validation, and performance analysis

The correctness, effectiveness, and efficiency of the business processes supported by the WFM system are vital to the organization. A workflow process definition which contains errors may lead to angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of a workflow definition may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to *analyze* a workflow process definition before it is put into production. Basically, there are three types of analysis:

- *validation*, i.e., testing whether the workflow behaves as expected,
- *verification*, i.e., establishing the correctness of a workflow, and
- *performance analysis*, i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization.

Validation can be done by interactive simulation: a number of fictitious cases are fed to the system to see whether they are handled well. For verification and performance analysis more advanced analysis techniques are needed. Fortunately, many powerful analysis techniques have been developed for Petri nets ([30, 63]). Linear algebraic techniques can be used to verify many properties, e.g., place invariants, transition invariants, and (non-)reachability. Coverability graph analysis, model checking, and reduction techniques can be used to analyze the dynamic behavior of a Petri net. Simulation and Markov-chain analysis can be used for performance evaluation (cf. [59, 63]). The abundance of available analysis techniques shows that Petri nets can be seen as a solver independent medium between the design of the workflow process definition and the analysis of the resulting workflow.

### 3.2 Verification of the control-flow perspective

In this tutorial we restrict ourselves to *workflow verification*, i.e., we will not discuss techniques for validation and performance analysis. Moreover, we restrict ourselves to the *control flow perspective*. Although each of the perspectives mentioned in Section 2.1 is relevant, the general focus of this tutorial is on control flow perspective, i.e., we use WF-nets as a starting point and demonstrate that Petri-net-based analysis techniques can be used to verify the correctness of a workflow process.

We abstract from the *resource perspective* because, given today's workflow technology, at any time there is only one resource working on a task which is being executed for a specific case. In today's WFM systems it is not possible to specify that several resources are collaborating in executing a task. Note that even if multiple persons are executing one task, e.g., writing a report, only one person is allocated to that task from the perspective of the WFM system: This is the person that selected the work item from the in-basket (i.e., the electronic worktray). Since a person is working on one task at a time and each task is eventually executed by one person (although it may be allocated to a group a people), it is sufficient to check whether all resource classes have at least one resource. In contrast to many other application domains such a flexible manufacturing systems, anomalies such as a deadlock resulting from locking problems are not possible. Therefore, from the viewpoint of verification, i.e., analyzing the logical correctness

of a workflow, it is reasonable to abstract from resources. However, if in the future collaborative features are explicitly supported by the workflow management system (i.e., a tight integration of groupware and workflow technology), then the resource perspective should be taken into account.

We partly abstract from the *data perspective*. The reason we abstract from production data is that these are outside the scope of the WFM system. These data can be changed at any time without notifying the WFM system. In fact their existence does not even depend upon the workflow application and they may be shared among different workflows, e.g., the bill-of-material in manufacturing is shared by production, procurement, sales, and quality control processes. The control data used by the WFM system to route cases are managed by the WFM system. However, some of these data are set or updated by humans or applications. For example, a decision is made by a manager based on intuition or a case is classified based on a complex calculation involving production data. Clearly, the behavior of a human or a complex application cannot be modeled completely. Therefore, some abstraction is needed to incorporate the data perspective when verifying a given workflow. The abstraction used in this section is the following. Since control data (i.e., workflow attributes such as the age of a customer, the department responsible, or the registration date) are only used for the routing of a case, we incorporate the routing decisions but not the actual data. For example, the decision to accept or to reject an insurance claim is taken into account, but not the actual data where this decision is based on. Therefore, we consider each choice to be a non-deterministic one. There are other reasons for abstracting from the workflow attributes. If we are able to prove soundness (i.e., the correctness criterion used in this section) for the situation without workflow attributes, it will also hold for the situation with workflow attributes (assuming certain fairness properties). Last but not least, we abstract from triggers and workflow attributes because it allows us to use ordinary Petri nets (i.e., P/T nets) rather than high-level Petri nets. From an analysis point of view, this is preferable because of the availability of efficient algorithms and powerful analysis tools.

For similar reasons we (partly) abstract from the *task* and *operation perspectives*. We consider tasks to be atomic and abstract from the execution of operations inside tasks. The WFM system can only launch applications or trigger people and monitor the results. It cannot control the actual execution of the task. Therefore, from the viewpoint of verification, it is reasonable to focus on the control-flow perspective. In fact, it suffices to consider the life cycle of one case in isolation. The only way cases interact directly is through the competition for resources and the sharing of production data. (Note that control data are strictly separated.) Therefore, if we abstract from resources and data, it suffices to consider one case in isolation. The competition between cases for resources is only relevant for performance analysis.

### 3.3 Soundness

In this section we summarize some of the basic results for WF-nets presented in [1, 3, 4].

The three requirements stated in Definition 11 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement which should be satisfied:

*For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net. These two additional requirements correspond to the so-called *soundness property*.

**Definition 12 (Sound).** *A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:*

*(i) For every state $M$ reachable from state $i$, there exists a firing sequence leading from state $M$ to state $o$. Formally:*[2]

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

*(ii) State $o$ is the only state reachable from state $i$ with at least one token in place $o$. Formally:*

$$\forall_M (i \xrightarrow{*} M \ \wedge \ M \geq o) \Rightarrow (M = o)$$

*(iii) There are no dead transitions in $(PN, i)$. Formally:*

$$\forall_{t \in T} \exists_{M, M'} \ i \xrightarrow{*} M \xrightarrow{t} M'$$

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 12 states that starting from the initial state (state *i*), it is always possible to reach the state with one token in place *o* (state *o*). If we assume a strong notion of fairness, then the first requirement implies that eventually state $o$ is reached. Strong fairness means in every infinite firing sequence, each transition fires infinitely often. The fairness assumption is reasonable in the context of WFM: All choices are made (implicitly or explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. Note that the traditional notions of fairness (i.e., weaker forms of fairness with just local conditions, e.g., if a transition is enabled infinitely often, it will fire eventually) are not sufficient. See [3, 53] for more details. The second requirement states that the moment a token is put in place *o*, all the other places should be empty. The last requirement states that there are no dead transitions (tasks) in the initial state *i*.

The WF-net shown in Figure 5 is sound. This can be verified by checking the three requirements stated in Definition 12. Note that Figure 5 shows triggers and uses syntactic sugaring. For verification we will abstract from this and consider the pure WF-net as shown in Figure 4.

Figure 6 shows a WF-net which is not sound. There are several deficiencies. If *time_out_1* and *processing_2* fire or *time_out_2* and *processing_1* fire, the WF-net will not terminate properly because a token gets stuck in *c4* or *c5*. If *time_out_1* and *time_out_2* fire, then the task *processing_NOK* will be executed twice and because of the presence of two tokens in *o* the moment of termination is not clear.

[2] Note that there is an overloading of notation: the symbol *i* is used to denote both the *place i* and the *state* with only one token in place *i* (see Section 2.2).

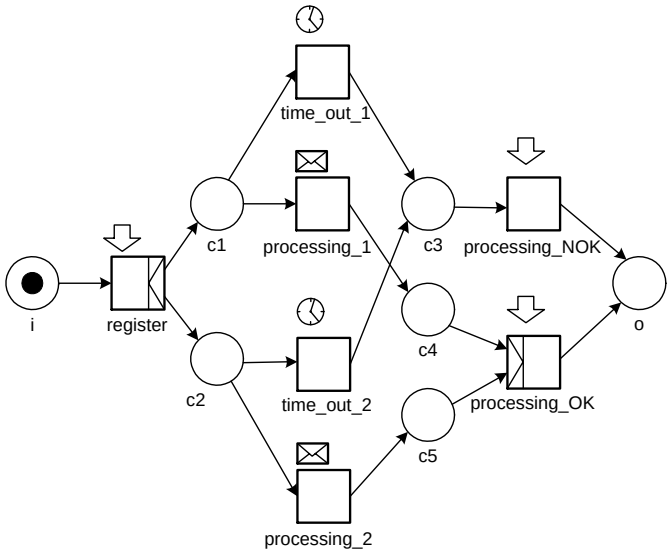


**Fig. 6.** Another WF-net for the processing of complaints.

Given a WF-net $PN = (P, T, F)$, we want to decide whether $PN$ is sound. In [1] we have shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained by adding an extra transition $t^*$ which connects $o$ and $i$. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{(o, t^*), (t^*, i)\}$. In the remainder we will call such an extended net the *short-circuited* net of $PN$. The short-circuited net allows for the formulation of the following theorem.

**Theorem 1.** *A WF-net $PN$ is sound if and only if $(\overline{PN}, i)$ is live and bounded.*

*Proof.* See [1]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness.

In literature there exist many variants of the "classical" notion of soundness used here. Juliane Dehnert uses the notion of relaxed soundness where proper termination is possible but not guaranteed [28, 34]. The main idea is that the scheduler of the workflow system should avoid problems like deadlocks etc. In [54] Ekkart Kindler et al. define variants of soundness tailored towards interorganizational workflows. Kees van Hee et al. [44] define a notion of soundness where multiple tokens in the source place are considered. A WF-net is $k$-sound if it "behaves well" when there are $k$ tokens in place $i$, i.e., no deadlocks and in the end there are $k$ tokens in place $o$. Robert van der Toorn uses the same concept in [71]. In [18, 7] stronger notions of soundness are used and places have to be safe. Another notion of soundness is used in [51, 52] where there is

not a single sink place but potentially multiple sink transitions. See [71] for the relation between these variants of the same concept. Other references using (variants of) the soundness property include [41, 60]. For simplicity we restrict ourselves to the classical notion of soundness defined in Definition 12.

### 3.4 Structural characterization of soundness

Theorem 1 gives a useful characterization of the quality of a workflow process definition. However, there are a number of problems:

- For a complex WF-net it may be intractable to decide soundness. (For arbitrary WF-nets liveness and boundedness are decidable but also EXPSPACE-hard, cf. Cheng, Esparza and Palsberg [26].)
- Soundness is a minimal requirement. Readability and maintainability issues are not addressed by Theorem 1.
- Theorem 1 does not show how a non-sound WF-net should be modified, i.e., it does not identify constructs which invalidate the soundness property.

These problems stem from the fact that the definition of soundness relates to the dynamics of a WF-net while the workflow designer is concerned with the static structure of the WF-net. Therefore, it is interesting to investigate structural characterizations of sound WF-nets. For this purpose we introduce three interesting subclasses of WF-nets: *free-choice WF-nets*, *well-structured WF-nets*, and *S-coverable WF-nets*.

**Free-Choice WF-Nets** Most of the WFM systems available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These WFM systems use building blocks such as the AND-split, AND-join, OR-split and OR-join to specify workflow procedures. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places. This means that for these WFM systems, a workflow procedure corresponds to a free-choice Petri net (cf. Definition 7).

It is easy to see that a process definition composed of AND-splits, AND-joins, OR-splits and OR-joins is free-choice. If two transitions $t_1$ and $t_2$ share an input place ($\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they are part of an OR-split, i.e., a "free choice" between a number of alternatives. Therefore, the sets of input places of $t_1$ and $t_2$ should match ($\bullet t_1 = \bullet t_2$). Figure 6 shows a free-choice WF-net. The WF-net shown in Figure 4 is not free-choice; *archive* and *reminder* share an input place but the two corresponding input sets differ.

We have evaluated many WFM systems and only some of these systems (e.g., COSA [66]) allow for a construct which is comparable to a non-free choice WF-net. Therefore, it makes sense to consider free-choice Petri nets in more detail. Clearly, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property. Another reason for restricting WF-nets to free-choice

Petri nets is the following. If we allow non-free-choice Petri nets, then the choice between conflicting tasks *may* be influenced by the order in which the preceding tasks are executed. The routing of a case should be independent of the order in which tasks are executed. A situation where the free-choice property is violated is often a mixture of parallelism and choice. Figure 7 shows such a situation. Firing transition *t1* introduces parallelism. Although there is no real choice between *t2* and *t5* (*t5* is not enabled), the parallel execution of *t2* and *t3* results in a situation where *t5* is not allowed to occur. However, if the execution of *t2* is delayed until *t3* has been executed, then there is a real choice between *t2* and *t5*. In our opinion parallelism itself should be separated from the choice between two or more alternatives. Therefore, we consider the non-free-choice construct shown in Figure 7 to be improper. In literature, the term *confusion* is often used to refer to the situation shown in Figure 7.

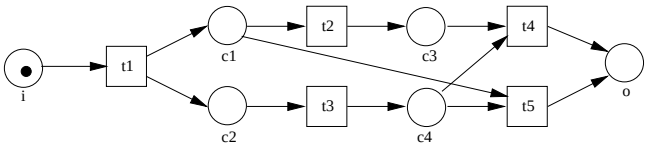


**Fig. 7.** A non-free-choice WF-net containing a mixture of parallelism and choice.

Free-choice Petri nets have been studied extensively [30, 29, 35, 43], because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. For example, the well-known Rank Theorem [30] enables us to formulate the following corollary.

**Corollary 1.** *The following problem can be solved in polynomial time.*
*Given a free-choice WF-net, to decide if it is sound.*

*Proof.* Let $PN$ be a free-choice WF-net. The short-circuited net $\overline{PN}$ is also free-choice. Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time (Rank Theorem [30]). By Theorem 1, this corresponds to soundness. □

Corollary 1 shows that, for free-choice nets, there are efficient algorithms to decide soundness. Moreover, a sound free-choice WF-net is guaranteed to be safe (given an initial state with just one token in $i$).

**Lemma 1.** *A sound free-choice WF-net is safe.*

*Proof.* Let $PN$ be a sound free-choice WF-net. $\overline{PN}$ is the Petri net $PN$ extended with a transition connecting $o$ and $i$. $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable [30], i.e., each place is part of an embedded strongly connected state-machine component. Since initially there is just one token $(\overline{PN}, i)$ is safe and so is $(PN, i)$. □

Safeness is a desirable property, because it makes no sense to have multiple tokens in a place representing a condition. A condition is either true (1 token) or false (no tokens).

Although most WFM systems only allow for free-choice workflows, free-choice WF-nets are not a completely satisfactory structural characterization of "good" workflows. On the one hand, there are non-free-choice WF-nets which correspond to sensible workflows (cf. Figure 4). On the other hand there are sound free-choice WF-nets which make no sense. Nevertheless, the free-choice property is a desirable property. If a workflow can be modeled as a free-choice WF-net, one should do so. A workflow specification based on a free-choice WF-net can be enacted by most workflow systems. Moreover, a free-choice WF-net allows for efficient analysis techniques and is easier to understand. Non-free-choice constructs such as the construct shown in Figure 7 are a potential source of anomalous behavior (e.g., deadlock) which is difficult to trace.

**Well-Structured WF-Nets** Another approach to obtain a structural characterization of "good" workflows, is to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. As shown in Figure 8, an AND-split should be complemented by an AND-join and an OR-split should be complemented by an OR-join.

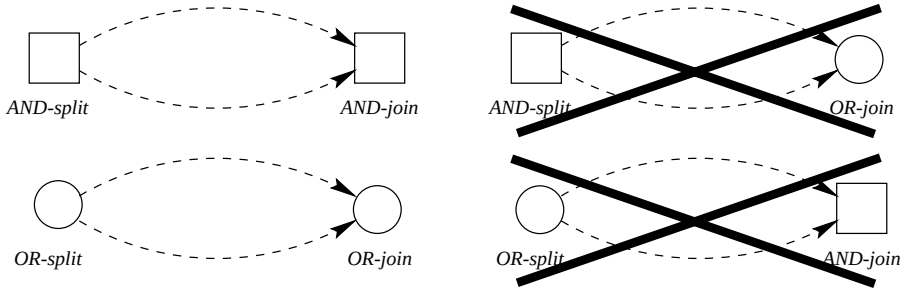


**Fig. 8.** Good and bad constructions.

One of the deficiencies of the WF-net shown in Figure 6 is the fact that the AND-split *register* is complemented by the OR-join *c3* or the OR-join *o*. To formalize the concept illustrated in Figure 8 we give the following definition.

**Definition 13 (Well-handled).** *A Petri net $PN$ is well-handled iff, for any pair of nodes $x$ and $y$ such that one of the nodes is a place and the other a transition and for any pair of elementary paths $C_1$ and $C_2$ leading from $x$ to $y$, $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.*

Note that the WF-net shown in Figure 6 is not well-handled. Well-handledness can be decided in polynomial time by applying a modified version of the max-flow min-cut technique. A Petri net which is well-handled has a number of nice properties, e.g., strong connectedness and well-formedness coincide.

**Lemma 2.** *A strongly connected well-handled Petri net is well-formed.*

*Proof.* Let $PN$ be a strongly connected well-handled Petri net. Clearly, there are no circuits that have PT-handles nor TP-handles [36]. Therefore, the net is structurally bounded (See Theorem 3.1 in [36]) and structurally live (See Theorem 3.2 in [36]). Hence, $PN$ is well-formed. □

Clearly, well-handledness is a desirable property for any WF-net $PN$. Moreover, we also require the short-circuited $\overline{PN}$ to be well-handled. We impose this additional requirement for the following reason. Suppose we want to use $PN$ as a part of a larger WF-net $PN'$. $PN'$ is the original WF-net extended with an "undo-task". See Figure 9. Transition $undo$ corresponds to the undo-task, transitions $t1$ and $t2$ have been added to make $PN'$ a WF-net. It is undesirable that transition $undo$ violates the well-handledness property of the original net. However, $PN'$ is well-handled iff $\overline{PN}$ is well-handled. Therefore, we require $\overline{PN}$ to be well-handled. We use the term *well-structured* to refer to WF-nets whose extension is well-handled.

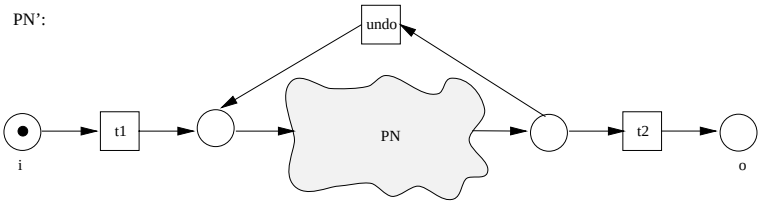


**Fig. 9.** The WF-net $PN'$ is well-handled iff $\overline{PN}$ is well-handled.

**Definition 14 (Well-structured).** *A WF-net $PN$ is well-structured iff $\overline{PN}$ is well-handled.*

Well-structured WF-nets have a number of desirable properties. Soundness can be verified in polynomial time and a sound well-structured WF-net is safe. To prove these properties we use some of the results obtained for *elementary extended non-self controlling nets*.

**Definition 15 (Elementary extended non-self controlling).** *A Petri net $PN$ is elementary extended non-self controlling (ENSC) iff, for every pair of transitions $t_1$ and $t_2$ such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, there does not exist an elementary path $C$ leading from $t_1$ to $t_2$ such that $\bullet t_1 \cap \alpha(C) = \emptyset$.*

**Theorem 2.** *Let $PN$ be a WF-net. If $PN$ is well-structured, then $\overline{PN}$ is elementary extended non-self controlling.*

*Proof.* Assume that $\overline{PN}$ is not elementary extended non-self controlling. This means that there is a pair of transitions $t_1$ and $t_k$ such that $\bullet t_1 \cap \bullet t_k \neq \emptyset$ and there exist an elementary path $C = \langle t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ leading from $t_1$ to $t_k$ and $\bullet t_1 \cap \alpha(C) = \emptyset$.

Let $p_1 \in {\bullet}t_1 \cap {\bullet}t_k$. $C_1 = \langle p_1, t_k \rangle$ and $C_2 = \langle p_1, t_1, p_2, t_2, \ldots, p_k, t_k \rangle$ are paths leading from $p_1$ to $t_k$. (Note that $C_2$ is the concatenation of $\langle p_1 \rangle$ and $C$.) Clearly, $C_1$ is elementary. We will also show that $C_2$ is elementary. $C$ is elementary, and $p_1 \notin \alpha(C)$ because $p_1 \in {\bullet}t_1$. Hence, $C_2$ is also elementary. Since $C_1$ and $C_2$ are both elementary paths, $C_1 \neq C_2$ and $\alpha(C_1) \cap \alpha(C_2) = \{p_1, t_k\}$, we conclude that $\overline{PN}$ is not well-handled. □

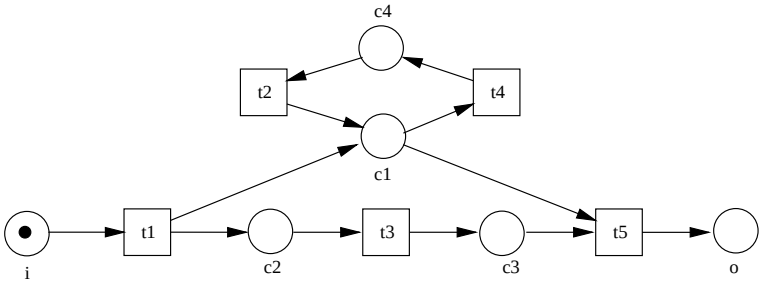


**Fig. 10.** A well-structured WF-net.

Consider for example the WF-net shown in Figure 10. The WF-net is well-structured and, therefore, also elementary extended non-self controlling. However, the net is not free-choice. Nevertheless, it is possible to verify soundness for such a WF-net very efficiently.

**Corollary 2.** *The following problem can be solved in polynomial time.*
*Given a well-structured WF-net, to decide if it is sound.*

*Proof.* Let $PN$ be a well-structured WF-net. The short-circuited net $\overline{PN}$ is elementary extended non-self controlling (Theorem 2) and structurally bounded (see proof of Lemma 2). For bounded elementary extended non-self controlling nets the problem of deciding whether a given marking is live, can be solved in polynomial time (See [23]). Therefore, the problem of deciding whether $(\overline{PN}, i)$ is live and bounded can be solved in polynomial time. By Theorem 1, this corresponds to soundness. □

**Lemma 3.** *A sound well-structured WF-net is safe.*

*Proof.* Let $\overline{PN}$ be the net $PN$ extended with a transition connecting $o$ and $i$. $\overline{PN}$ is extended non-self controlling. $\overline{PN}$ is covered by state-machines (S-components), see Corollary 5.3 in [23]. Hence, $\overline{PN}$ is safe and so is $PN$ (see proof of Lemma 1). □

Well-structured WF-nets and free-choice WF-nets have similar properties. In both cases soundness can be verified very efficiently and soundness implies safeness. In spite of these similarities, there are sound well-structured WF-nets which are not free-choice (Figure 10) and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net which is neither free-choice nor well-structured (Figures 4 and 7).

**S-Coverable WF-Nets** What about the sound WF-nets shown in Figure 4 and Figure 7? The WF-net shown in Figure 7 can be transformed into a free-choice well-structured WF-net by separating choice and parallelism. The WF-net shown in Figure 4 cannot be transformed into a free-choice or well-structured WF-net without yielding a much more complex WF-net. Place *c7* acts as some kind of milestone which is tested by the task *reminder*. Traditional WFM systems which do not make the state of the case explicit, are not able to handle the workflow specified by Figure 4. Only WFM systems such as COSA [66] have the capability to enact such a state-based workflow. Nevertheless, it is interesting to consider generalizations of free-choice and well-structured WF-nets: *S-coverable WF-nets* can be seen as such a generalization.

**Definition 16 (S-coverable).** *A WF-net $PN$ is S-coverable if the short-circuited net $\overline{PN}$ is S-coverable.*

The WF-nets shown in Figure 4 and Figure 7 are S-coverable. The WF-net shown in Figure 6 is not S-coverable. The following two corollaries show that S-coverability is a generalization of the free-choice property and well-structuredness.

**Corollary 3.** *A sound free-choice WF-net is S-coverable.*

*Proof.* The short-circuited net $\overline{PN}$ is free-choice and well-formed. Hence, $\overline{PN}$ is S-coverable (cf. [30]). □

**Corollary 4.** *A sound well-structured WF-net is S-coverable.*

*Proof.* $\overline{PN}$ is extended non-self controlling (Theorem 2). Hence, $\overline{PN}$ is S-coverable (cf. Corollary 5.3 in [23]). □

All the sound WF-nets presented in this tutorial are S-coverable. Every S-coverable WF-net is safe. The only WF-net which is not sound, i.e., the WF-net shown in Figure 6, is not S-coverable. These and other examples indicate that there is a high correlation between S-coverability and soundness. It seems that S-coverability is one of the basic requirements any workflow process definition should satisfy. From a formal point of view, it is possible to construct WF-nets which are sound but not S-coverable. Typically, these nets contain places which do not restrict the firing of a transition, but which are not in any S-component. (See for example Figure 65 in [62].) From a practical point of view, these WF-nets are to be avoided. WF-nets which are not S-coverable are difficult to interpret because the structural and dynamical properties do not match. For example, these nets can be live and bounded but not structurally bounded. There seems to be no practical need for using constructs which violate the S-coverability property. Therefore, we consider S-coverability to be a basic requirement any WF-net should satisfy.

Another way of looking at S-coverability is the following interpretation: S-components corresponds to *document flows*. To handle a workflow several pieces of information are created, used, and updated. One can think of these pieces of information as physical documents, i.e., at any point in time the document is in one place in the WF-net. Naturally, the information in one document can be copied to another document while executing a task (i.e., transition) processing both documents. Initially, all documents are present but a document can be empty (i.e., corresponds to a blank piece

paper). It is easy to see that the flow of one such document corresponds a state machine (assuming the existence of a transition $t^*$). These document flows synchronize via joint tasks. Therefore, the composition of these flows yields an S-coverable WF-net. One can think of the document flows as threads. Consider for example the short-circuited net of the WF-net shown in Figure 4. This net can be composed out of the following two threads: (1) a thread corresponding to logistic subprocess (places $start$, $c0$, $c1$, $c3$, $c4$, $c5$, $c7$, and $end$) and (2) a thread corresponding to the actual processing of the complaint (places $start$, $c2$, $c6$, $c8$, and $end$). Note that the tasks *register* and *archive* are used in both threads.

Although a WF-net can, in principle, have exponentially many S-components, they are quite easy to compute for workflows encountered in practice (see also the above interpretation of S-component as document flows or threads). Note that S-coverability only depends on the structure and the degree of connectedness is generally low (i.e., the incidence matrix of a WF-net typically has few non-zero entries). Unfortunately, in general, it is not possible to verify soundness of an S-coverable WF-net in polynomial time. The problem of deciding soundness for an S-coverable WF-net is PSPACE-complete. For most applications this is not a real problem. Typically, the number of tasks in one workflow process definition is less than 100 and the number of states is less than half a million. Tools using standard techniques such as the construction of the coverability graph have no problems in coping with these workflow process definitions.

**Using the three structural characterizations** The three structural characterizations (free-choice, well-structured and S-coverable) turn out to be very useful for the analysis of workflow process definitions. Based on our experience, we have good reasons to believe that S-coverability is a desirable property any workflow definition should satisfy. Constructs violating S-coverability can be detected easily and tools can be build to help the designer to construct an S-coverable WF-net. S-coverability is a generalization of well-structuredness and the free-choice property (Corollary 3 and 4). Both well-structuredness and the free-choice property also correspond to desirable properties of a workflow. A WF-net satisfying at least one one of these two properties can be analyzed very efficiently. However, we have shown that there are workflows that are not free-choice and not well-structured. Consider for example Figure 4. The fact that task *register* tests whether there is a token in *c5*, prevents the WF-net from being free-choice or well-structured. Although this is a very sensible workflow, most WFM systems do not support such an advanced routing construct. Even if one is able to use state-based workflows (e.g., COSA) allowing for constructs which violate well-structuredness and the free-choice property, then the structural characterizations are still useful. If a WF-net is not free-choice or not well-structured, one should locate the source which violates one of these properties and check whether it is really necessary to use a non-free-choice or a non-well-structured construct. If the non-free-choice or non-well-structured construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors. This way the readability and maintainability of a workflow process definition can be improved.

### 3.5 Woflan

Few tools aiming at the verification of workflow processes exist. Woflan [73, 72] and Flowmake [64] are two notable exceptions. We have been working on Woflan since 1997. Figure 11 shows a screenshot of Woflan. Woflan combines state-of-the-art scientific results with practical applications [73, 72]. Woflan can interface with leading WFM systems such as Staffware, MQSeries Workflow and COSA but also PNML [24]. It can also interface with BPR-tools such as Protos. Workflow processes designed using any of these tools can be verified for correctness. It turns out that the challenge is not to decide whether the design is sound or not. The real challenge is to provide diagnostic information that guides the designer to the error. Woflan also supports the inheritance notions mentioned before. Given two workflow designs, Woflan is able to decide whether one is a subclass of the other. Tools such as Woflan illustrate the benefits of a more fundamental approach.

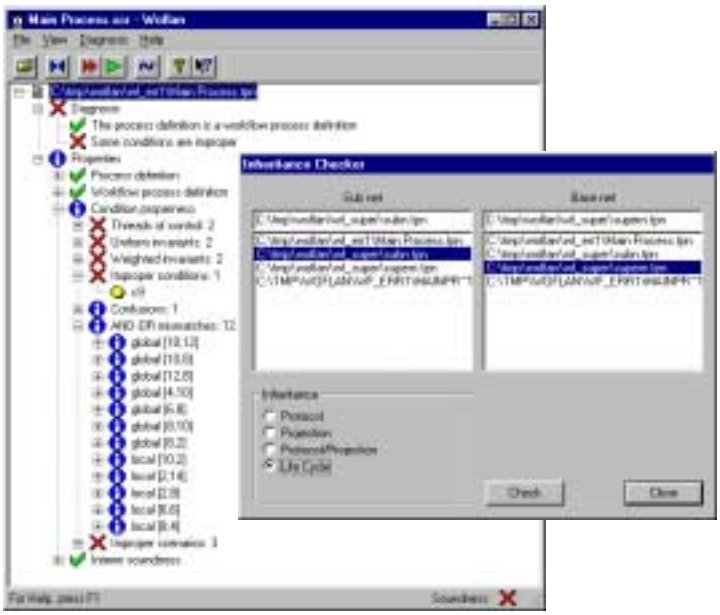


**Fig. 11.** A screenshot showing the verification and validation capabilities of Woflan.

### 3.6 Exercise

Consider the solution of the exercise given in Section 2.4. Verify whether the WF-net is sound and make sure that there is an S-cover. A simple verification "web service" is provided via `http://is.tm.tue.nl/research/woflan/`. This web service uses Woflan to verify whether a given process model is sound. Use this web service or download Woflan to check the correctness of your solution.

# 4 Workflow management systems

In this section we provide insight into the functionality of existing WFM systems. First we provide an overview of the workflow market. Then we introduce the typical architecture of a WFM system, followed by an example of a concrete system. Again, we conclude the section with an exercise.

## 4.1 Overview

In Section 1 we put WFM in a historical perspective and using Figure 2 we discussed the maturity of the BPM market. At this point in time hundreds of WFM/BPM products are available. To illustrate this we use two diagrams of Michael Zur Muehlen [61]. Figure 12 gives a historic overview of office automation and workflow prototypes [61]. Figure 13 provides a historic overview of commercial WFM systems. These two figures show that: (1) workflow management is not something that started in the nineties but already in the seventies with the work of Ellis (*OfficeTalk*, [32]) and Zisman (*Scoop*,[78]) and (2) the number of commercial systems has considerably grown in recent years. Note that given the dynamics of the workflow market, it is difficult to keep diagrams like the one shown in Figure 13 up-to-date. For example, Figure 13 does not show recent systems like FLOWer [22]. Moreover, systems are often named different for commercial reasons. For example, IBM's MQSeries Workflow (formerly known as FlowMark) was recently renamed into WebSphere MQ Workflow.

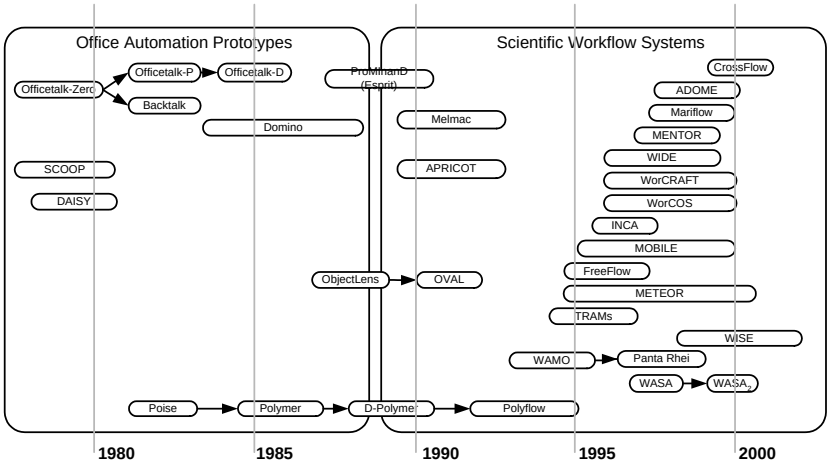


**Fig. 12.** Historic overview of early systems and research prototypes (Taken from [61]).

Unfortunately, figures 12 and 13 do not show the increased maturity of WFM/BPM products. It also does not show that products target at different types of processes. A well-know classification of WFM systems is given in [40] where the authors distinguish

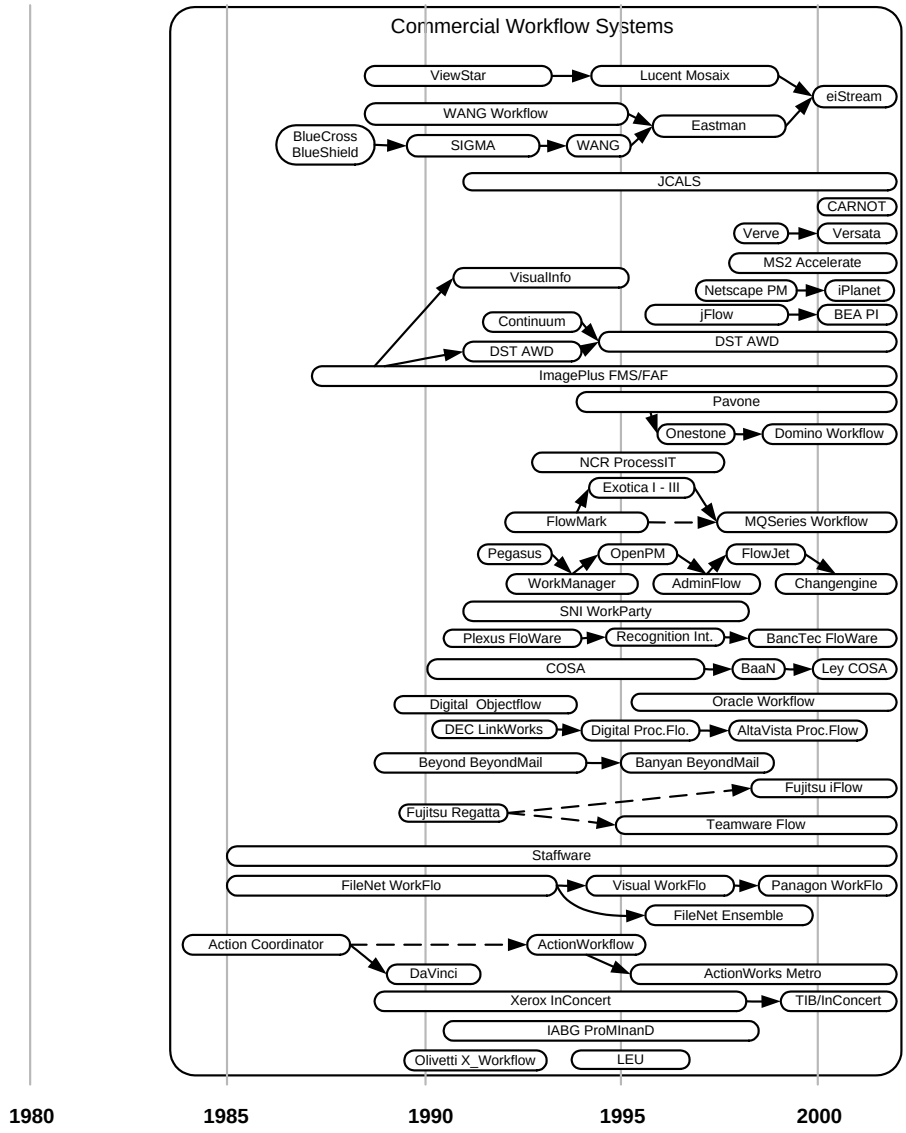


**Fig. 13.** Historic overview of commercial workflow management systems (Taken from [61]).

between ad-hoc, administrative, and production workflows and discuss the continuum from human-oriented to system-oriented WFM systems. However, we prefer to use the more recent classification shown in Figure 14 to describe the “workflow spectrum”.

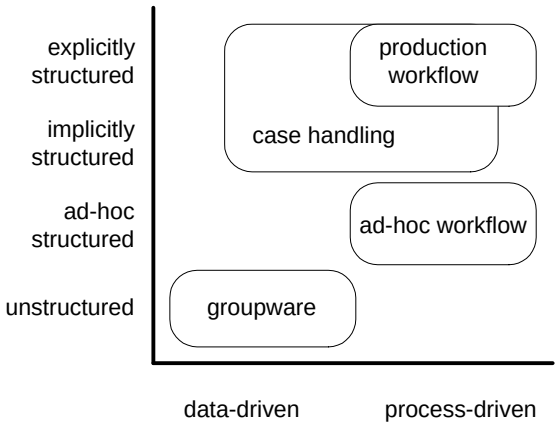


**Fig. 14.** Classification of systems to support work processes.

Figure 14 shows four types of systems: groupware, production workflow, ad-hoc workflow, and case-handling systems. These systems are characterized in terms of their “focus” (data-driven, process driven, or both) and their “degree of structuredness”. Traditional groupware products like Lotos Notes and MS Exchange and production workflow systems like Staffware and MQSeries Workflow form two ends of a spectrum. As Figure 14 shows, traditional groupware products are data-driven (focus on the sharing of information rather than the process) and support only unstructured processes. Note that Lotus Notes and Exchange are not “process-aware” (unless components like Domino Workflow are added). Production workflow systems are process-aware and aim at structured processes. In order to enact a workflow using a production workflow system one needs to explicitly specify all possible routes. If something is not explicitly specified at design time, it is not possible. Ad-hoc WFM systems like InConcert (TIBCO), Ensemble (Filenet), and TeamWARE Flow (TeamWARE Group) allow for the creation and modification of workflow processes at execution time. Each case has a private process model and therefore the traditional problems encountered when changing a workflow specification can be avoided. Ad-hoc WFM systems allow for a lot of flexibility. The WFM system InConcert even allows the user to initiate a case having an empty process model. When the case is handled, the workflow model is extended to reflect the work conducted. Another possibility is to start using a template. The moment a case is initiated, the corresponding process model is instantiated using a template. After instantiation, the case has a private copy of the template, which can be modified while the process is running. InConcert also supports “workflow design by discovery”: The routing of any completed workflow instance can be used to create a new template. This way actual workflow executions can be used to create workflow process defini-

tions. Figure 14 shows that ad-hoc workflow management systems like InConcert are process-driven and ad-hoc structured. Case-handling systems like FLOWer and Vectus can be positioned in-between groupware, production workflow, and ad-hoc workflow. Unlike in ad-hoc workflow systems the end-users are not expected to change or create process models. Instead the following paradigms are used for case handling [8]:

- avoid context tunneling by providing all information available (i.e., present the case as a whole rather than showing just bits and pieces),
- decide which activities are enabled on the basis of the information available rather than the activities already executed,
- separate work distribution from authorization and allow for additional types of roles, not just the execute role,
- allow workers to view and add/modify data before or after the corresponding activities have been executed (e.g., information can be registered the moment it becomes available).

For more information on case handling we refer to [8, 22]. Clearly the classification of systems is not as clear-cut as Figure 14 may suggest. Lotus Notes can be extended with Domino Workflow to join groupware and production workflow functionalities. Staffware Case Handler and the COSA Activity Manager are extensions of production workflow systems in the direction of case handling (both are based on the generic solution of BPi).

In this tutorial we focus on the classical production workflow systems. However, it is important to understand that they are part of a spectrum and that their application is limited to a specific type of processes (process-driven and explicitly structured).

### 4.2 Architecture

As indicated by Figure 14, WFM systems target at different processes. Therefore, it is not surprising that there is not one architecture that "fits all systems". Therefore, we present the so-called *reference model* of the Workflow Management Coalition (WfMC) [38, 55]. Figure 15 shows an overview of this reference model. The reference model describes the major components and interfaces within a workflow architecture. The core of any workflow system is the *workflow enactment service*. The workflow enactment service provides the run-time environment which takes care of the control and execution of the workflow. For technical or managerial reasons the workflow enactment service may use multiple *workflow engines*. A workflow engine handles selected parts of the workflow and manages selected parts of the resources. The *process definition tools* are used to specify and analyze workflow process definitions and/or resource classifications. These tools are used at design time. In most cases, the process definition tools can also be used as a BPR-toolset. Most WFM systems provide three process definition tools: (1) a tool with a graphical interface to define workflow processes, (2) a tool to specify resource classes (organizational model), and (3) a simulation tool to analyze a specified workflow.[3] The end-user communicates with the workflow system

[3] In many cases simulation is offered through some export to a standard simulation tool, e.g., COSA supports simulation through an export to ExSpect.

via the *workflow client applications*. An example of a workflow client application is the well-known *in-basket*. Via such an in-basket work items are offered to the end user. By selecting a work item, the user can execute a task for a specific case. If necessary, the workflow engine invokes applications via Interface 3. The *administration and monitoring tools* are used to monitor and control the workflow. These tools are used to register the progress of cases and to detect bottlenecks. Moreover, these tools are also used to set parameters, allocate people and handle abnormalities. Via Interface 4 the workflow system can be connected to other workflow systems. To standardize the five interfaces shown in Figure 15, the WfMC aims at a common *Workflow Application Programming Interface* (WAPI). The WAPI is envisaged as a common set of API calls and related interchange formats which may be grouped together to support each of the five interfaces (cf. [55]). In Section 5.2 we will describe XPDL, the XML-based language suggested by the WfMC to exchange process definition (i.e., a concrete language for Interface 1).

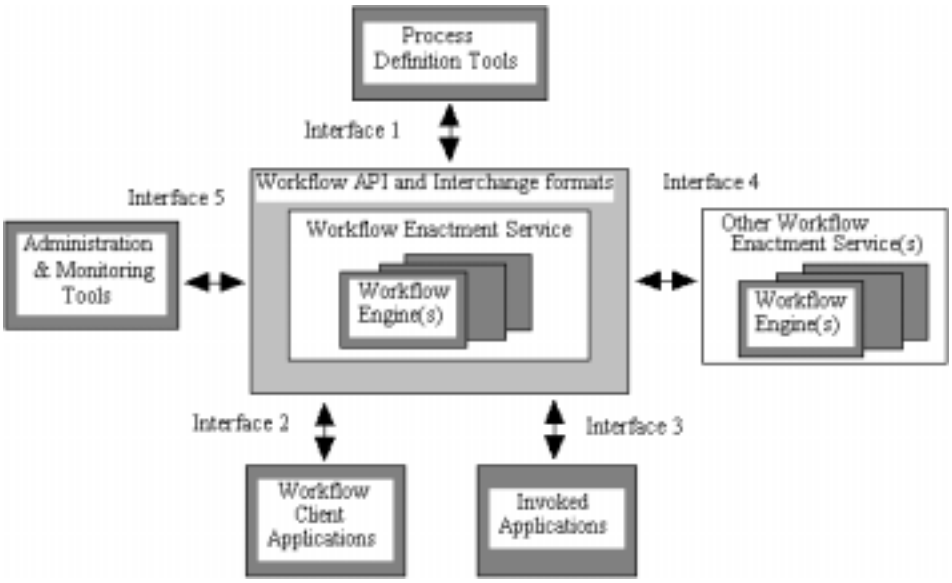


**Fig. 15.** Reference model of the Workflow Management Coalition (WfMC).

### 4.3 Example of a WFM system: Staffware

As indicated in Section 4.1, many WFM systems are available. In this tutorial we only show one system in more detail. Staffware is one of the most widespread WFM systems in the world. In 1998, it was estimated by the Gartner Group that Staffware has 25 percent of the global market [25]. Staffware provides the functionality described in the reference model shown in Figure 15. Figure 16 shows some screenshots of the Staffware product. The top window shows the design tool of Staffware while defining a simple

workflow process. Work is offered through so-called work queues. One worker can have multiple work queues and one work queue can be shared among multiple workers. The window in the middle shows the set of available work queues (left) and the content of one of these work queues (right). The bottom window shows an audit trail of a case. The three windows show only some of the capabilities offered by Staffware. It is fairly straightforward to map these windows onto the architecture shown in Figure 15.

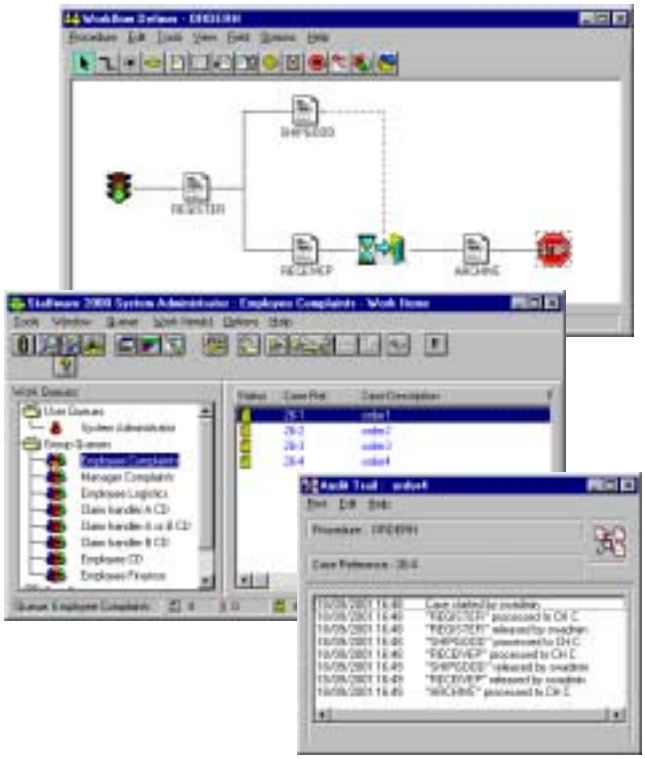

**Fig. 16.** The Graphical Workflow Definer, Work Queue, and Audit Trail of Staffware.

Let us now consider the modeling language used by Staffware. In Staffware, tasks are called *steps*. There are several kinds of steps: *automatic steps* (offered to an application instead of an end-user), *normal steps* (executed by an end-user), and *event steps* (triggered by some external event). The semantics of a step are OR-join/AND-split, i.e., a step becomes enabled if one of the preceding steps is completed and the completion of step will trigger all subsequent steps. Since the OR-join/AND-split semantics is fixed, two additional building blocks are needed: the *wait step* and the *condition*. The wait step can be used to synchronize flows and has AND-join/AND-split semantics. To model choices, i.e., OR-splits, the condition building block can be used. Staffware only allows for binary choices, i.e., just two possible outcomes (e.g., YES and NO). Staffware processes always start with a start step which is denoted by a symbol rep-

resenting a traffic light. Termination in Staffware is implicit, i.e., it is possible to start multiple parallel threads which end concurrently. Therefore, there is no need to have one sink node representing the completion of a case. The end of a thread is denoted by a stop symbol. Conditions are modeled by diamond shaped symbols. Wait steps are modeled by symbols in the shape of a sand timer. The basic semantics of a step, a condition, and a wait are shown in Figure 17.[4]

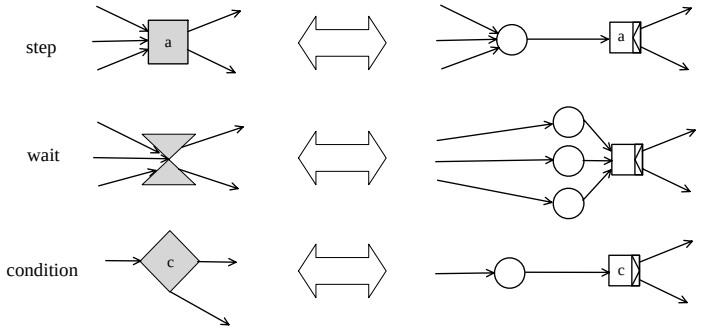


**Fig. 17.** The semantics of some of the Staffware constructs (left) expressed in Petri nets (right).

Using this translation shown in Figure 17, it is straightforward to map the Staffware model shown in Figure 16 onto a WF-net. The result is shown in Figure 18.

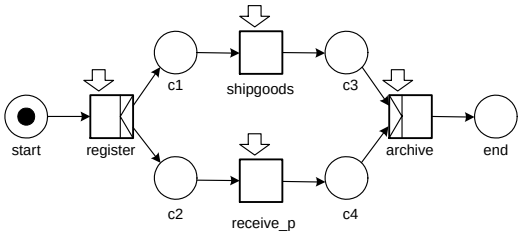


**Fig. 18.** The Staffware model shown in Figure 16 expressed in terms of a WF-net.

Let us consider now a larger Staffware model also including advanced concepts like time triggers and multiple ending points. For this purpose, we use the model shown in Figure 19. It models a simplified workflow in a travel agency. To organize a trip, a travel agency executes several tasks. First the customer is registered. Then an employee searches for opportunities which are communicated to the customer. Then the customer will be contacted to find out whether she or he is still interested in the trip of this

[4] Note that the semantics of Staffware steps include a number of particularities not included in the mapping, cf. [72].

agency and whether more alternatives are desired. There are three possibilities: (1) the customer is not interested at all, (2) the customer would like to see more alternatives, and (3) the customer selects an opportunity. If the customer selects a trip, the trip is booked. In parallel one or two types of insurance are prepared if they are desired. A customer can take insurance for trip cancellation or/and for baggage loss. Note that a customer can decide not to take any insurance, just trip cancellation insurance, just baggage loss insurance, or both types of insurance. Two weeks before the start date of the trip the documents are sent to the customer. A trip can be cancelled at any time after completing the booking process (including the insurance) and before the start date. Note that customers who are not insured for trip cancellation can cancel the trip (but will get no refund). Most of the model is self-explanatory. The two OR-join symbols represent “dummy tasks”, i.e., Staffware steps not implementing any real task. For the cancellation two steps with a time-out are used: *CANCEL* and *CANCEL2*. The clock symbol is used to indicate steps with a time-out. In such as step, the lower branch is taken if the step is not executed within a given period. For simplicity we did not model all triggers and simplified the choice for both types of insurances.

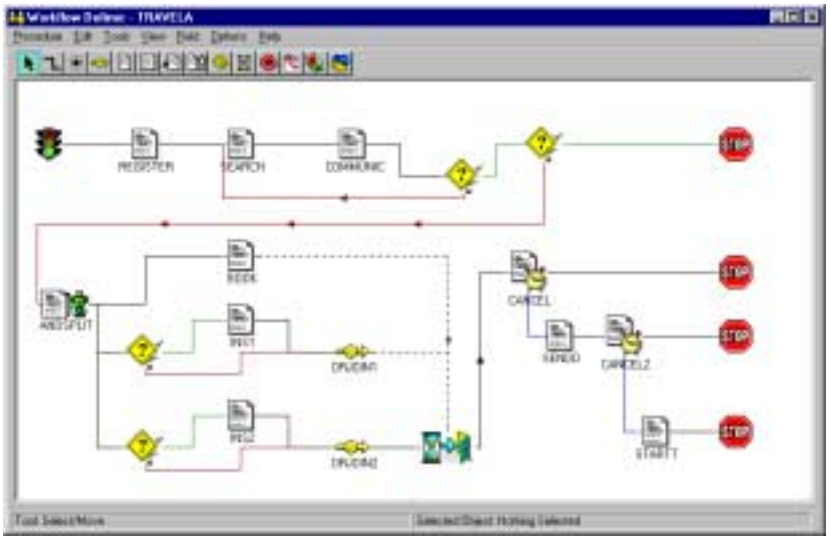

**Fig. 19.** The workflow of a travel agency modeled in terms of the Staffware language.

Let us now translate the model shown in Figure 19 into a WF-net. We do not use the Staffware names but the names used in the original description (Staffware only allows names of up-to 8 characters). The WF-net shown in Figure 20, like any WF-net, has a source place which serves as the start condition (i.e., case creation) and a sink place which serves as the end condition (i.e., case completion). First, the tasks *register*, *search*, *communicate*, and *contact_cust* are executed sequentially. Task *contact_cust* is an OR-split with three possible outcomes: (1) the customer is not interested at all, i.e.,

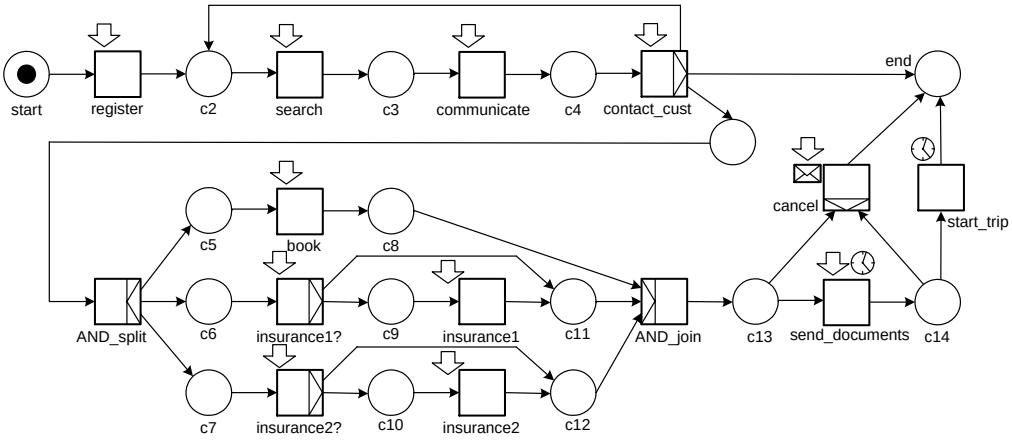


**Fig. 20.** The travel agency's workflow expressed in terms of a WF-net.

a token is put into *end*, (2) the customer would like to see more alternatives, i.e., a token is put into *c2*, and (3) the customer selects an opportunity, i.e., a token is put into *c15* to initiate the booking. Tasks *AND_split* and *AND_join* have just been added for routing purposes. These routing tasks enable the parallel execution of the booking and insurance tasks. The task *book* corresponds to the actual booking of the trip. Tasks *insurance1* and *insurance2* correspond to handling both types of insurance. Since both types of insurance are optional, there is a bypass for each of these tasks. The OR-split *insurance1?* allows for a bypass of task *insurance1* by putting a token in *c11*. After handling the booking and optional insurances the AND-join puts a token in *c13*. The remainder of the process is, from the viewpoint of triggers, very interesting. Note that all tasks executed before this point are either tasks that require a resource trigger or automatic tasks added for routing purposes only. The downward facing arrows denote the resource triggers. If the case is in *c13*, then the normal flow of execution is to first execute task *send_documents* and then execute *start_trip*. Note that task *send_documents* requires both a resource trigger and a time trigger. These two triggers indicate that two weeks before the beginning of the trip a worker sends the documents to the customer. Task *start_trip* has been added for routing purposes and requires a time trigger. Without task *start_trip*, i.e., putting the token in *end* after sending the documents, it would have been impossible to cancel the trip after sending the documents. Task *cancel* is an explicit OR-join and requires both a resource trigger and an external trigger. This task is only executed if it is triggered by the customer. Task cancel can only be executed if the case is in *c13* or *c14*, i.e., after handling the booking and insurance related tasks and before the trip starts.

It is interesting to compare figures 19 and 20. Although the WF-net has more symbols because of the explicit modeling of states (i.e., places), it seems to be a more direct and more elegant way to model the process.

### 4.4 Exercises

To conclude this section, we provide two small exercises.

- Consider the Staffware process shown in Figure 21. Translate this Staffware model into a WF-net.
- Consider the WF-net shown in Figure 5. Translate this WF-net into a Staffware model, i.e., provide a diagram like the one shown in Figure 21 for the order processing process given in Section 2.3.

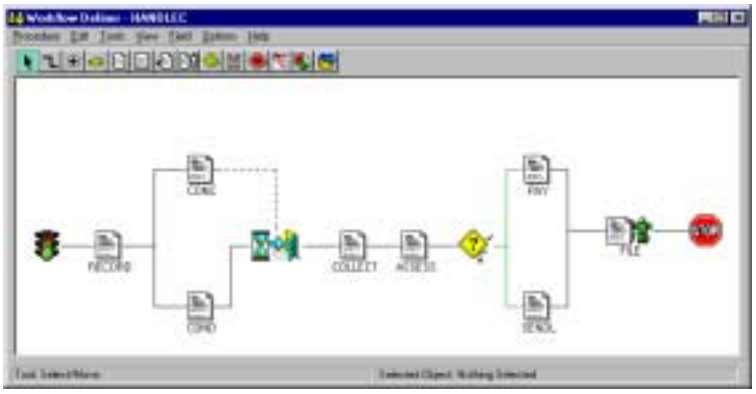

**Fig. 21.** Exercise: translate this Staffware process into a WF-net.

A solution of the first exercise is given in [12]. A solution for the second exercise is not given and is also far from trivial given the fact that Staffware does not support the Milestone and Deferred choice patterns (cf. Section 5.1 and [15]).

## 5 Workflow standards: An evaluation of XPDL based on its support for workflow patterns

There are many languages and standards in the WFM/BPM domain. It is impossible to discuss all of these in any detail. Instead we focus on a single standard. XPDL is not the most important standard but it is typical for many other standards and propriety languages of workflow vendors. For a critical evaluation of XPDL, we use the set of workflow patterns described in [15]. The remainder of this section is structured as follows. First, we introduce the 20 workflow patterns used to evaluate XPDL. Then, in Section 5.2, we provide an overview of the XPDL language. In Section 5.3 the language is analyzed using the 20 workflow patterns. Section 5.4 discusses one of the core semantical problems: The join construct. Finally, Section 5.5 concludes the section by comparing XPDL with WFM systems and other standards such as UML Activity Diagrams, BPEL4WS, BPML, WSFL, XLANG, and WSCI.

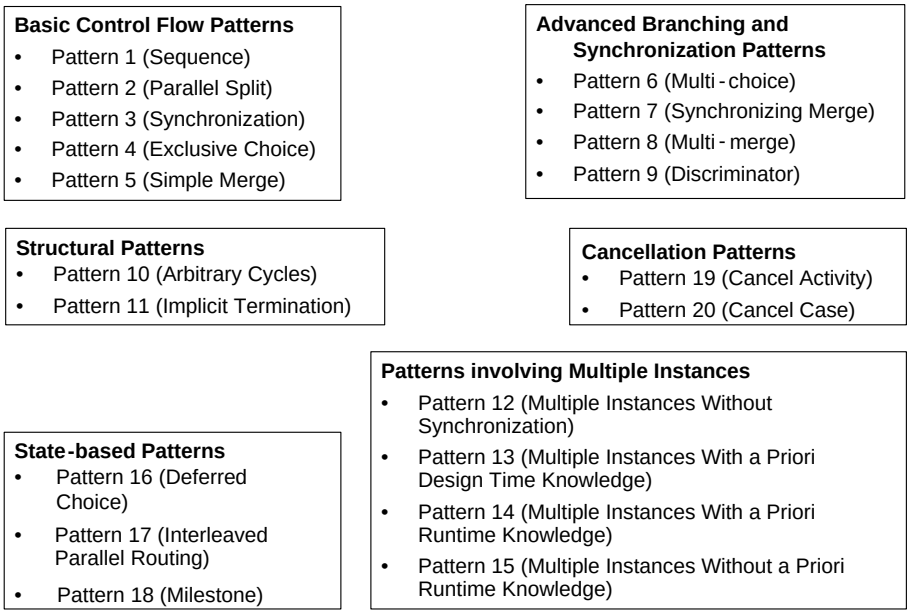


**Fig. 22.** Overview of the 20 workflow patterns described in [15].

### 5.1 Workflow patterns

Since 1999 we have been working on collecting a comprehensive set of workflow patterns [15]. The results have been made available through the "Workflow patterns WWW site" `http://www.workflowpatterns.com`. The patterns range from very simple patterns such as sequential routing (Pattern 1) to complex patterns involving complex synchronizations such as the discriminator pattern (Pattern 9). In this tutorial, we restrict ourselves to the 20 most relevant patterns. These patterns can be classified into six categories:

1. *Basic control-flow patterns.* These are the basic constructs present in most workflow languages to model sequential, parallel and conditional routing.
2. *Advanced branching and synchronization patterns.* These patterns transcend the basic patterns to allow for more advanced types of splitting and joining behavior. An example is the Synchronizing merge (Pattern 7) which behaves like an AND-join or XOR-join depending on the context.
3. *Structural patterns.* In programming languages a block structure which clearly identifies entry and exit points is quite natural. In graphical languages allowing for parallelism such a requirement is often considered to be too restrictive. Therefore, we have identified patterns that allow for a less rigid structure.
4. *Patterns involving multiple instances.* Within the context of a single case (i.e., workflow instance) sometimes parts of the process need to be instantiated multiple times, e.g., within the context of an insurance claim, multiple witness statements need to be processed.

5. *State-based patterns.* Typical workflow systems focus only on activities and events and not on states. This limits the expressiveness of the workflow language because it is not possible to have state dependent patterns such as the Milestone pattern (Pattern 18).
6. *Cancellation patterns.* The occurrence of an event (e.g., a customer canceling an order) may lead to the cancellation of activities. In some scenarios such events can even cause the withdrawal of the whole case.

Figure 22 shows an overview of the 20 patterns grouped into the six categories. A detailed discussion of these patterns is outside the scope of this tutorial. The interested reader is referred to [15] and `http://www.workflowpatterns.com`.

We have used these patterns to compare the functionality of numerous WFM systems. The result of this evaluation reveals that (1) the expressive power of contemporary systems leaves much to be desired and (2) the systems support different patterns. Note that we do not use the term "expressiveness" in the traditional or formal sense. If one abstracts from capacity constraints, any workflow language is Turing complete. Therefore, it makes no sense to compare these languages using formal notions of expressiveness. Instead we use a more intuitive notion of expressiveness which takes the modeling effort into account. This more intuitive notion is often referred to as suitability. See [51, 52] for a discussion on the distinction between formal expressiveness and suitability.

The observation that the expressive power of the available WFM systems leaves much to be desired, triggered the question: *How about XPDL as a workflow language?*

### 5.2 XPDL: XML Process Definition Language

The Workflow Management Coalition (WfMC) was founded in August 1993 as a international non-profit organization. Today there are about 300 members ranging from workflow vendors and users to analysts and university/research groups. The mission of the WfMC is to promote and develop the use of workflow through the establishment of standards for workflow terminology, interoperability and connectivity between workflow products. The WfMC's reference model identifies five interfaces, as shown in Section 4.2. One of the main activities since 1993 has been the development of standards for these interfaces. Interface 1 is the link between the so-called "Process Definition Tools" and the "Enactment Service" (cf. Figure 15). The Process Definition Tools are used to design workflows while the Enactment Service can execute workflows. The primary goal of Interface 1 is the import and export of process definitions. The WfMC defines a process definition as "The representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a WFM system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc." [55]. Clearly, there is a need for process definition interchange. First of all, within the context of a single workflow management system there has to be a connection between the design tool and the execution/run-time environment. Second, there may be the desire to use another design tool, e.g., a modeling tool like ARIS or Protos. Third, for analysis purposes it may be desirable to link the design tool to analysis software such as simulation

and verification tools. Fourth, the use of repositories with workflow processes requires a standardized language. Fifth, there may be the need to transfer a definition interchange from one engine to another.

To support the interchange of workflow process definitions, there has to be a standardized language [12, 38, 48, 55, 57, 58]. The WfMC started working on such a language soon after it was founded. This resulted in the Workflow Process Definition Language (WPDL) [74] presented in 1999. Although many vendors claimed to be WfMC compliant, few made a serious effort to support this language. At the same time, XML emerged as a standard for data interchange. Since WPDL was not XML-based, the WfMC started working a new language named XML Process Definition Language (XPDL). The starting point for XPDL was WPDL. However, XPDL should not be considered the XML version of WPDL. Several concepts have been added/changed and the WfMC remains fuzzy about the exact relationship between XPDL and WPDL. In October 2002, the WfMC released a "Final Draft" of XPDL [75].

In [75], the authors state "More complex transitions, which cannot be expressed using the simple elementary transition and the split and join functions associated with the from- and to- activities, are formed using dummy activities, which can be specified as intermediate steps between real activities allowing additional combinations of split and/or join operations. *Using the basic transition entity plus dummy activities, routing structures of arbitrary complexity can be specified.* Since several different approaches to transition control exist within the industry, several conformance classes are specified within XPDL. These are described later in the document." The sentence "Using the basic transition entity plus dummy activities, routing structures of *arbitrary complexity* can be specified." triggered us to look into the expressive power of XPDL.

XPDL [75] uses an XML-based syntax, specified by an XML schema. The main elements of the language are: Package, Application, WorkflowProcess, Activity, Transition, Participant, DataField, and DataType. The Package element is the container holding the other elements. The Application element is used to specify the applications/tools invoked by the workflow processes defined in a package. The element WorkflowProcess is used to define workflow processes or parts of workflow processes. A WorkflowProcess is composed of elements of type Activity and Transition. The Activity element is the basic building block of a workflow process definition. Elements of type Activity are connected through elements of type Transition. There are three types of activities: Route, Implementation, and BlockActivity. Activities of type Route are dummy activities just used for routing purposes. Activities of type BlockActivity are used to execute sets of smaller activities. Element ActivitySet refers to a self contained set of activities and transitions. A BlockActivity executes such an ActivitySet. Activities of type Implementation are steps in the process which are implemented by manual procedures (No), implemented by one of more applications (Tool), or implemented by another workflow process (Subflow). The Participant element is used to specify the participants in the workflow, i.e., the entities that can execute work. There are 6 types of participants: ResourceSet, Resource, Role, OrganizationalUnit, Human, and System. Elements of type DataField and DataType are used to specify workflow relevant data. Data is used to make decisions or to refer to data outside of the workflow, and is passed between activities and subflows.

In this section, we focus on the control-flow perspective. Therefore, we will not consider functionality related to the Package, Application, and Participant elements. Moreover, we will only consider workflow relevant data from the perspective of routing. Appendix A shows selected parts of the XPDL Schema [75] relevant for this tutorial. The listing shows the elements Activity, TransitionRestriction, TransitionRestrictions, Join, Split, Transition and Condition. An activity may have one of more "transition restrictions" to specify the split/join behavior. If there is a transition restriction of type Join, the restriction is either set to AND or to XOR. The WfMC defines the semantics of such a restriction as follows: "AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s)." and "XOR: Join for alternative threads: No synchronization is required." [75]. Similarly, there are transition restrictions of type Split that are set to either AND or XOR with the following semantics: "AND: Defines a number of possible concurrent threads represented by the outgoing Transitions of this Activity. If the Transitions have conditions the actual number of executed parallel threads is dependent on the conditions associated with each transition, which are evaluated concurrently." and "XOR: List of Identifiers of outgoing Transitions of this Activity, representing. Alternatively executed transitions. The decision as to which single transition route is selected is dependent on the conditions of each individual transition as they are evaluated in the sequence specified in the list. If an unconditional Transition is evaluated or transition with condition OTHERWISE this ends the list evaluation." [75]. Appendix A also shows the definition of element Transition. A transition connects two activities as indicated by the From and To field and may contain a Condition element.

The WfMC acknowledges the fact that workflow languages use different styles and paradigms. To accommodate this, XPDL allows for vendor specific extensions of the language. In addition, XPDL distinguishes three conformance classes: non-blocked, loop-blocked, and full-blocked. These conformance classes refer to the network structure of a process definition, i.e., the graph of activities (nodes) and transitions (arcs). For conformance class non-blocked there are no restrictions. For conformance class loop-blocked the network structure has to be acyclic and for conformance class full-blocked there has to be a one-to-one correspondence between splits and joins of the same type. These conformance classes correspond to different styles of modeling. Graph based workflow languages like COSA and Staffware correspond to conformance class non-blocked. Languages such as MQSeries, WSFL, and BPEL4WS correspond to conformance class loop-blocked and block-structured languages such as XLANG are full-blocked.

A detailed introduction to XPDL is beyond the scope of this tutorial. For more details we refer to [75].

### 5.3 The Workflow Patterns in XPDL

In this section, we consider the 20 workflow patterns discussed in Section 5.1, and we show how and to what extent these patterns can be captured in XPDL. In particular, we indicate whether the pattern is directly supported by a XPDL construct. If this is not the

case, we sketch a workaround solution. Most of the solutions are presented in a simplified XPDL notation which is intended to capture the key ideas of the solutions while avoiding coding details. In other words, the fragments of XPDL definitions provided here are not “ready to be run”.

**WP1 Sequence** An activity in a workflow process is enabled after the completion of another activity in the same process. **Example:** After the activity *order registration* the activity *customer notification* is executed.

**Solution, WP1** This pattern is directly supported by the XPDL as illustrated in in Listing 1. Within the process Sequence two activities A and B are linked through transition AB.

**Listing 1 (Sequence)**

```
<WorkflowProcess Id="Sequence">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
            ...
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B"/>
    </Transitions>
</WorkflowProcess>
```

**WP2 Parallel Split** A point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order [55, 37]. **Example:** After activity *new cell phone subscription order* the activity *insert new subscription* in Home Location Registry application and *insert new subscription* in Mobile answer application are executed in parallel.

**WP3 Synchronization** A point in the process where multiple parallel branches converge into one single thread of control, thus synchronizing multiple threads [55]. It is an assumption of this pattern that after an incoming branch has been completed, it cannot be completed again while the merge is still waiting for other branches to be completed. Also, it is assumed that the threads to be synchronized belong to the same global process instance (i.e., to the same “case” in workflow terminology). **Example:** Activity *archive* is executed after the completion of both activity *send tickets* and activity *receive payment*. Obviously, the synchronization occurs within a single global process instance: the *send tickets* and *receive payment* must relate to the same client request.

**Listing 2 (Parallel Split/Synchronization)**

```
<WorkflowProcess Id="Parallel">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
              ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="AND">
                        <TransitionRefs>
                            <TransitionRef Id="B"/>
                            <TransitionRef Id="C"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
        <Activity Id="C">
            ....
        </Activity>
        <Activity Id="D">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Join Type="AND"/>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B"/>
        <Transition Id="AC" From="A" To="C"/>
        <Transition Id="BD" From="B" To="D"/>
        <Transition Id="CD" From="C" To="D"/>
    </Transitions>
</WorkflowProcess>
```

**Solutions, WP2 & WP3** This pattern directly supported by the XPDL. This is illustrated by the example shown in Listing 2. Within the process Parallel four activities are linked trough four transitions. Transitions AB and AC link the initial activity A to the two parallel activities B and C. Note that the split in activity A is of type AND and no transition conditions are specified. Transitions BD and CD link the two parallel activities B and C to the final activity D. Note that the join in activity D is of type AND and again no transition conditions are specified.

**WP4 Exclusive Choice** A point in the process where, based on a decision or workflow control data, one of several branches is chosen. **Example:** The manager is informed if an order exceeds $600, otherwise not.

**WP5 Simple Merge** A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel with another one (if it is not the case, then see the patterns Multi Merge and Discriminator). **Example:** After the payment is received or the credit is granted the car is delivered to the customer.

**Solutions, WP4 & WP5** XPDL can address the Exclusive choice pattern (WP4) in two ways. In both cases, an activity has a split and multiple outgoing transitions. One way is to use a split of type XOR, i.e., the first transition which as no condition or a condition which evaluates to true is taken. Another way is to use split of type AND and define mutual exclusive transition conditions. Listing 3 shows a solution using the first alternative. Listing 4 shows a solution using the second alternative. In the second solution transitions AB and AC have a condition. In the first solution transitions AB and AC do not have a condition which effectively implies that always the first one (AB) is taken. Besides normal conditions based on workflow relevant data, it is also possible to use conditions of type OTHERWISE (for the default branch to be taken if all other conditions evaluate to false) and of type EXCEPTION (for specifying the branch to be taken after an exception was raised). Listings 3 and 4 also show the direct support for the Simple merge (WP5).

**WP6 Multi-Choice** A point in the process, where, based on a decision or control data, a number of branches are chosen and executed as parallel threads. **Example:** After executing the activity *evaluate damage* the activity *contact fire department* or the activity *contact insurance company* is executed. At least one of these activities is executed. However, it is also possible that both need to be executed.

**Solution, WP6** XPDL provides direct support for the Multi-Choice pattern as shown in Listing 5. Depending on the value of amount activity B and/or C is/are executed, e.g., if the value of amount is 8 both activities are executed, otherwise just B (amount$> 5$) or C (amount$< 10$).

**WP7 Synchronizing Merge** A point in the process where multiple paths converge into one single thread. Some of these paths are "active" (i.e. they are being executed) and some are not. If only one path is active, the activity after the merge is triggered as soon

**Listing 3 (Exclusive Choice/Simple Merge)**

```
<WorkflowProcess Id="Choice1">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="XOR">
                        <TransitionRefs>
                            <TransitionRef Id="AB"/>
                            <TransitionRef Id="AC"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
        <Activity Id="C">
              ...
        </Activity>
        <Activity Id="D">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Join Type="XOR"/>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B"/>
        <Transition Id="AC" From="A" To="C"/>
        <Transition Id="BD" From="B" To="D"/>
        <Transition Id="CD" From="C" To="D"/>
    </Transitions>
</WorkflowProcess>
```

**Listing 4 (Exclusive Choice/Simple Merge)**

```
<WorkflowProcess Id="Choice2">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="AND">
                        <TransitionRefs>
                            <TransitionRef Id="AB"/>
                            <TransitionRef Id="AC"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
        <Activity Id="C">
            ...
        </Activity>
        <Activity Id="D">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Join Type="XOR"/>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B">
            <Condition Type="CONDITION">
                choice == "B" </Condition>
        </Transition>
        <Transition Id="AC" From="A" To="C">
            <Condition Type="CONDITION">
                choice == "C" </Condition>
        </Transition>
        <Transition Id="BD" From="B" To="D"/>
        <Transition Id="CD" From="C" To="D"/>
    </Transitions>
</WorkflowProcess>
```

**Listing 5 (Multi Choice/Synchronizing merge)**

```
<WorkflowProcess Id="Multi-choice">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="AND">
                        <TransitionRefs>
                            <TransitionRef Id="AB"/>
                            <TransitionRef Id="AC"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
        <Activity Id="C">
            ...
        </Activity>
        <Activity Id="D">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Join Type="AND"/>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B">
            <Condition Type="CONDITION">
                amount &gt; 5 </Condition>
        </Transition>
        <Transition Id="AC" From="A" To="C">
            <Condition Type="CONDITION">
                amount &lt; 10 </Condition>
        </Transition>
        <Transition Id="BD" From="B" To="D"/>
        <Transition Id="CD" From="C" To="D"/>
    </Transitions>
</WorkflowProcess>
```

as this path completes. If more than one path is active, synchronization of all active paths needs to take place before the next activity is triggered. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete. **Example:** After either or both of the activities *contact fire department* and *contact insurance company* have been completed (depending on whether they were executed at all), the activity submit report needs to be performed (exactly once).

**Solutions, WP7** According to [75] XPDL provides direct support for the Synchronizing merge pattern. Recall the definition of the AND restriction: "AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s)." [75] which suggests direct support for the Synchronizing merge pattern. If this is indeed the case, then Listing 5 indeed shows an example where activity D either merges or synchronizes the two ingoing transitions depending on the number of threads activated by activity A. Unfortunately, few workflow systems that claim to support XPDL have indeed this behavior. Moreover, XPDL allows for multiple interpretations as discussed in Section 5.4.

**WP8 Multi-Merge** A point in a process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for every action of every incoming branch. **Example:** Sometimes two or more branches share the same ending. Two activities *audit application* and *process applications* are running in parallel which should both be followed by an activity *close case*, which should be executed twice if the activities *audit application* and *process applications* are both executed.

**Solution, WP8** XPDL only allows for two types of joins: AND and XOR. The semantics of these two joins is not completely clear. A join of type XOR will offer the Simple merge pattern. Recall that the simple merge assumes that precisely one of the incoming transitions will occur. However, XPDL allows for situations where the more incoming transitions will or may occur. Consider Listing 6. Both B and C are executed. Since activity D has a join of type XOR it can already occur when one of these two have been executed. However, it is not clear how many times activity D will occur (and when). In [75] is is stated that "The XOR join initiates the Activity when the transition conditions of any (one) of the incoming transitions evaluates true.". Since it is not specified what should happen if multiple incoming transitions evaluate to true at the same time, we conclude that XPDL does not support the Multi-Merge (WP8). See [15] for typical work-arounds.

**WP9 Discriminator** A point in the workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important otherwise it could not really be used in the context of a loop). **Example:** To improve query response time a complex search is sent to two different databases over

**Listing 6 (Multi-merge?)**

```
<WorkflowProcess Id="Parallel">
    <ProcessHeader DurationUnit="Y"/>
    <Activities>
        <Activity Id="A">
              ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="AND">
                        <TransitionRefs>
                            <TransitionRef Id="B"/>
                            <TransitionRef Id="C"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
        <Activity Id="B">
            ...
        </Activity>
        <Activity Id="C">
            ....
        </Activity>
        <Activity Id="D">
            ...
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Join Type="XOR"/>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </Activities>
    <Transitions>
        <Transition Id="AB" From="A" To="B"/>
        <Transition Id="AC" From="A" To="C"/>
        <Transition Id="BD" From="B" To="D"/>
        <Transition Id="CD" From="C" To="D"/>
    </Transitions>
</WorkflowProcess>
```

the Internet. The first one that comes up with the result should proceed the flow. The second result is ignored.

**Solution, WP9** XPDL allows for situations where multiple incoming transitions will or may occur. However, the precise semantics of a join of type XOR is not specified and, similar to WP8, we conclude that the Discriminator (WP9) is not supported.

**WP10 Arbitrary Cycles** A point where a portion of the process (including one or more activities and connectors) needs to be "visited" repeatedly *without imposing restrictions* on the number, location, and nesting of these points. Note that block-oriented languages and languages providing constructs such as "while do", "repeat until" typically impose such restrictions, e.g., it is not possible to jump from one loop into another loop.

**Solution, WP10** XPDL distinguishes three conformance classes: non-blocked, loop-blocked, and full-blocked. Conformance class "non-blocked" directly supports this pattern. Note that the transitions basically define a relation and allow for any graph including cyclic ones. For the other conformance classes this is not allowed. For conformance class loop-blocked the network structure has to be acyclic and for conformance class full-blocked there has to be a one-to-one correspondence between splits and joins of the same type.

**WP11 Implicit Termination** A given subprocess is terminated when there is nothing left to do, i.e., termination does not require an explicit termination activity. The goal of this pattern is to avoid having to join divergent branches into a single point of termination.

**Solution, WP11** XPDL, assuming conformance class "non-blocked", allows for arbitrary graph-like structures. As a result it is possible to have multiple activities without input transitions (i.e., source activities) and multiple activities without output transitions (sink activities). The latter suggests direct support for WP11. Unfortunately, [75] does not clarify the semantics of XPDL in the presence of multiple source and sink activities, e.g., Do all source activities need to be executed or just one? Although XPDL does not specify the expected behavior in such cases, we give it the benefit of the doubt. Note that this illustrates that conformance is still ill-defined in [75] since it refers to syntax rather than semantics.

**WP12 MI without Synchronization** Within the context of a single case, multiple instances of an activity may be created, i.e. there is a facility for spawning off new threads of control, all of them independent of each other. The instances might be created consecutively, but they will be able to run in parallel, which distinguishes this pattern from the pattern for Arbitrary Cycles. **Example:** When booking a trip, the activity *book flight* is executed multiple times if the trip involves multiple flights.

**Solution, WP12** An activity may be refined into a subflow. The subflow may be executed synchronously or asynchronously. In case of asynchronous execution, the activity

is continued after an instance of the subflow is initiated. This way it is possible to "spawn-off" subflows and thus realizing WP12.

**WP13-WP15 MI with Synchronization** A point in a workflow where a number of instances of a given activity are initiated, and these instances are later synchronized, before proceeding with the rest of the process. In WP13 the number of instances to be started/synchronized is known at design time. In WP14 the number is known at some stage during run time, but before the initiation of the instances has started. In WP15 the number of instances to be created is not known in advance: new instances are created on demand, until no more instances are required. **Example of WP15:** When booking a trip, the activity *book flight* is executed multiple times if the trip involves multiple flights. Once all bookings are made, an invoice is sent to the client. How many bookings are made is only known at runtime through interaction with the user (or with an external process).

**Solutions, WP13-WP15** If the number of instances to be synchronized is known at design time (WP13), a simple solution is to replicate the activity as many times as it needs to be instantiated, and run the replicas in parallel. Therefore, WP13 is supported. However, it is clear that there is no direct support for WP14 and WP15 because any solution will involve explicit bookkeeping of the number of active instances. In fact in [75] is is stated that "Synchronization with the initiated subflow, if required, has to be done by other means such as events, not described in this document." when describing the functionality of asynchronous subflows. Therefore, we conclude that there is no support for WP14 and WP15. Again we refer to [15] for typical workarounds.

**WP16 Deferred Choice** A point in a process where one among several alternative branches is chosen based on information which is not necessarily available when this point is reached. This differs from the normal exclusive choice, in that the choice is not made immediately when the point is reached, but instead several alternatives are offered, and the choice between them is delayed until the occurrence of some event. **Example:** When a contract is finalized, it has to be reviewed and signed either by the director or by the operations manager, whoever is available first. Both the director and the operations manager would be notified that the contract is to be reviewed: the first one who is available will proceed with the review.

**Solution, WP16** XPDL only allows for choices resulting from conditions on transitions. Hence each choice is directly-based on workflow relevant data and it is not possible offer the choice to the environment. XPDL does not allow for the definition of states (like places in a Petri net) nor constructs like the choice construct in BPML and WSCI and the pick construct in XLANG and BPEL4WS. There is no simple work-around for this omission since it is not possible to shift the moment of decision from the end of an activity to the start of an activity. Moreover, XPDL does not allow for the specification of triggers and/or external events.

**WP17 Interleaved Parallel Routing** A set of activities is executed in an arbitrary order. Each activity in the set is executed exactly once. The order between the activities is

decided at run-time: it is not until one activity is completed that the decision on what to do next is taken. In any case, no two activities in the set can be active at the same time. **Example:** At the end of each year, a bank executes two activities for each account: *add interest* and *charge credit card costs*. These activities can be executed in any order. However, since they both update the account, they cannot be executed at the same time.

**Solution, WP17** Since XPDL does not allow for the definition of states, it is not possible to enforce some kind of mutual exclusion. Hence there is no support for WP17. Even the work-arounds described in [15] are difficult, if not impossible, to apply.

**WP18 Milestone** A given activity can only be enabled if a certain milestone has been reached which has not yet expired. A milestone is defined as a point in the process where a given activity has finished and another activity following it has not yet started. **Example:** After having placed a purchase order, a customer can withdraw it at any time before the shipping takes place. To withdraw an order, the customer must complete a withdrawal request form, and this request must be approved by a customer service representative. The execution of the activity *approve order withdrawal* must therefore follow the activity *request withdrawal*, and can only be done if: (i) the activity *place order* is completed, and (ii) the activity *ship order* has not yet started.

**Solution, WP18** XPDL does not provide a direct support for capturing this pattern. Therefore, a work-around solution has to be used. Again it is difficult to construct solutions inspired by the ideas in [15]. Since other patterns like WP16 and WP19 are not supported, potential solutions lead to complex process definitions for simply checking the state in a parallel branch.

**WP19 Cancel Activity & WP20 Cancel Case** A cancel activity terminates a running instance of an activity, while canceling a case leads to the removal of an entire workflow instance. **Example of WP19:** A customer cancels a request for information. **Example of WP20:** A customer withdraws his/her order.

**Solutions, WP19 & WP20** XPDL does not provide explicit constructs for WP19 and WP20. The concept of exceptions seems to be related, but like many other concepts ill-defined. The only construct in XPDL that can raise an exception is the deadline element. Deadlines are used to raise an exception upon the expiration of a specific period of time. A deadline can be raised synchronously or asynchronously: “If the deadline is synchronous, then the activity is terminated before flow continues on the exception path.” and “If the deadline is asynchronous, then an implicit AND-SPLIT is performed, and a new thread of processing is started on the appropriate exception transition.” [75]. An exception may trigger a transition but cannot be used to cancel activities or cases. Hence, XPDL does not support WP19 and WP20.

### 5.4 Many ways to join

In this section, we evaluated XPDL with respect to the patterns. A more detailed analysis reveals that, not only does XPDL have problems with respect to several patterns, the

semantics of many constructs is unclear. To illustrate this we focus on transition restrictions of type Join. The restriction is either set to AND or to XOR and the WfMC defines these settings as follows: "AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s)." and "XOR: Join for alternative threads: No synchronization is required." [75]. To demonstrate that these descriptions do not fully specify the intended behavior, Figure 23 shows seven possible interpretations each expressed in terms of a Petri net (some extended with inhibitor arcs, cf. [63]). Note that Petri nets have formal semantics, and thus, Figure 23 fully specifies the behavior of each construct. Also note that we restrict ourselves to local constructs, i.e., the there are no dependencies other than on the activities directly connected to the join.

The first two constructs correspond to the most straightforward interpretations of the AND-join (Figure 23(a)) and XOR-join (Figure 23(b)). In Figure 23(a), activity C always synchronizes A and B, i.e., if A is never executed, C is never executed.[5] In Figure 23(b), activity C is executed once for each occurrence of A and B. Although Figure 23(a) and Figure 23(b) seem to correspond to straightforward interpretations of the AND-join and XOR-join, few WFM systems actually exhibit this behavior. The other constructs in Figure 23 show other interpretations for both the AND-join and/or XOR-join encountered in contemporary systems. Figure 23(c) shows the situation where activity A is blocked if C was not executed since the last occurrence of A. Similarly, activity B is blocked if C was not executed since the last occurrence of B. Note that this construct uses two inhibitor arcs (i.e., the two connections involving a small circle). Unlike a normal directed arc in Petri net, an inhibitor arc models the requirement that a place has to be empty, i.e., A is only enabled if the input place (not shown) contains a token *and* the output place is empty. Figure 23(d) shows a similar construct but now for the XOR-join, i.e., both activity A and activity B are blocked if C was not executed since the last occurrence of A or B. The WFM system COSA [67] uses this interpretation for the AND-join and XOR-join. Figures 23 (c) and (d) use inhibitor arcs to make sure that activity C is only enabled once. This is realized by blocking the preceding activities if needed. An alternative approach is to simply remove additional tokens. Figure 23(e) shows an approach where C synchronizes both flows if both A and B have been executed. If only one of them has been executed, there is no synchronization. Note that there are three instances of C: one for the situation where only A was executed, one for situation where both A and B have been executed, and one where only B was executed. The two inhibitor arcs make sure that the two flows are synchronized if possible. Figure 23(f) shows a similar, but slightly different, approach where simply every attempt to enable C for the second time is ignored. If C is already enabled, then the right transition will occur, otherwise the left one. Consider the scenario where A occurs twice before execution C. In Figure 23(e), C will be executed twice, while in Figure 23(f) C will be executed only once. Many systems have a behavior similar to Figure 23(e)/(f), e.g., a normal step in Staffware [69] behaves as indicated by Figure 23(f). (See [72] for a more detailed analysis of Staffware steps.) Although widely supported, the interpretation given in Figure 23(e)/(f) is not very desirable from a modeling point of view

[5] Note that this is not the case in XPDL.

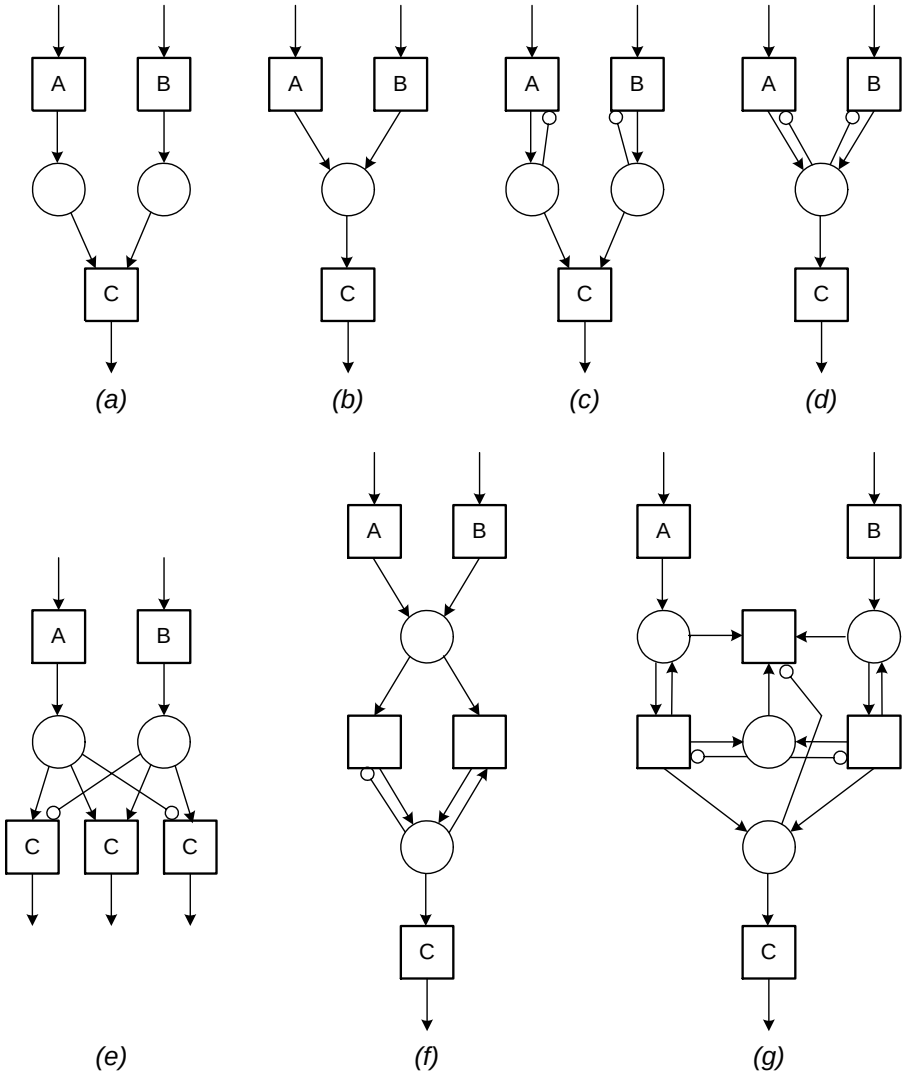


**Fig. 23.** Seven frequently used ways to join two flows (expressed in terms of Petri nets with inhibitor arcs [63]).

since it introduces "race conditions", e.g., the number of times C is executed depends on the interleaving of A, B, and C activities. Figure 23(g) gives yet another interpretation of the AND/XOR-join. C is enabled immediately after the first occurrence of A or B, but after it occurs it is blocked until the other activity has also been executed, i.e., the construct is reset once each of A, B, and C has occurred. Note that this interpretation corresponds to WP9 (Discriminator pattern).

Figure 23 shows that there are many ways to join two flows. In fact, there are many more interpretations. An example is the so-called "wait step" in Staffware [69] which only synchronizes the first time if it is put in a loop (see [72] for more details). Another example is the join in IBM's MQSeries Workflow [46], BPEL4WS [27], and WSFL (Web Services Flow Language, [56]) which decides whether it has to synchronize or not based on the so-called "Dead-Path-Elimination (DPE)" [57]. Given the quote "AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s)." in [75], the latter interpretation seems to be closest to XPDL. Unfortunately, other than IBM-influenced products and standards, no other vendors are using nor supporting this interpretation since it does not allow for Arbitrary cycles (WP10) [9].

The dilemma of joining mixtures of alternative or parallel flows has been discussed in scientific literature. See [9] for pointers to related papers and an elaborate discussion in the context of Event-driven Process Chains (EPC's).

The fact that there are many ways to join and that in [75] the WfMC leaves room for multiple interpretations, brings us to the issue of *conformance*. In [75] it is stated that "A product that claims conformance must generate valid, syntactically correct XPDL, and must be able to read all valid XPDL.". Unfortunately, this quote, but also the rest of [75], does not address the issue of semantics. Note that it is rather easy to generate and read valid XPDL files. The difficult part is to be able to interpret XPDL generated by another tool and execute the workflow as intended.

### 5.5 Comparing XPDL with other languages and standards

Thus far, we provided a critical evaluation of XPDL based on a set of 20 basic workflow patterns. To conclude this section, we compare XPDL with other standards and 15 workflow products.

Table 1 shows an evaluation of XPDL and six other standards. If a standard directly supports the pattern through one of its constructs, it is rated +. If the pattern is not *directly* supported, it is rated +/-. Any solution which results in spaghetti diagrams or

[6] Although the description of the AND-join suggests support for WP7, XPDL does not specify its precise behavior. In fact, for conformance class "non-blocked", it is unclear how WP7 could be supported

[7] For conformance class "non-blocked", arbitrary graph-like structures are allowed, including arbitrary cycles. For the other conformance classes this is explicitly excluded.

[8] For all conformance classes there may be multiple source and/or sink activities. Hence, from a syntactical point of view WP11 is supported. Unfortunately, no semantics are given for this construct.

| pattern | standard | | | | | | |
|---|---|---|---|---|---|---|---|
| | XPDL | UML | BPEL4WS | BPML | XLANG | WSFL | WSCI |
| 1 (seq) | + | + | + | + | + | + | + |
| 2 (par-spl) | + | + | + | + | + | + | + |
| 3 (synch) | + | + | + | + | + | + | + |
| 4 (ex-ch) | + | + | + | + | + | + | + |
| 5 (simple-m) | + | + | + | + | + | + | + |
| 6 (m-choice) | + | - | + | - | - | + | - |
| 7 (sync-m) | +[6] | - | + | - | - | + | - |
| 8 (multi-m) | - | - | - | +/- | - | - | +/- |
| 9 (disc) | - | - | - | - | - | - | - |
| 10 (arb-c) | +[7] | - | - | - | - | - | - |
| 11 (impl-t) | +[8] | - | + | + | - | + | + |
| 12 (mi-no-s) | + | - | + | + | + | + | + |
| 13 (mi-dt) | + | + | + | + | + | + | + |
| 14 (mi-rt) | - | + | - | - | - | - | - |
| 15 (mi-no) | - | - | - | - | - | - | - |
| 16 (def-c) | - | + | + | + | + | - | + |
| 17 (int-par) | - | - | +/- | - | - | - | - |
| 18 (milest) | - | - | - | - | - | - | - |
| 19 (can-a) | - | + | + | + | + | + | + |
| 20 (can-c) | - | + | + | + | + | + | + |

**Table 1.** A comparison of XPDL with other standards such as UML Activity Diagrams, BPEL4WS, BPML, XLANG, WSFL, and WSCI.

coding, is considered as giving no direct support and is rated -. The rating of XPDL is as explained in this section.

UML activity diagrams [42] are intended to model both computational and organizational processes. Increasingly, UML activity diagrams are also used for workflow modeling. Therefore, it is interesting to analyze their expressiveness using the set of basic workflow patterns as shown in the table. for more information see [31].

The recently released BPEL4WS (Business Process Execution Language for Web Services, [27]) specification builds on IBM's WSFL (Web Services Flow Language, [56]) and Microsoft's XLANG [70]. XLANG is a block-structured language with basic control flow structures such as sequence, switch (for conditional routing), while (for looping), all (for parallel routing), and pick (for race conditions based on timing or external triggers). In contrast to XLANG, WSFL is not limited to block structures and allows for directed graphs. The graphs can be nested but need to be acyclic. Iteration is only supported through exit conditions, i.e., an activity/subprocess is iterated until its exit condition is met. The control flow part of WSFL is almost identical to the workflow language used by IBM's MQ Series Workflow. See [76, 77] for more information about the evaluation of BPEL4WS, XLANG, and WSFL using the patterns.

BPML (Business Process Modeling language, [21]) is a standard developed and promoted by BPMI.org (the Business Process Management Initiative). BPMI.org is supported by several organizations, including Intalio, SAP, Sun, and Versata. The Web Service Choreography Interface (WSCI, [20]) submitted in June 2002 to the W3C by

BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation, and Sun Microsystems. There is a substantial overlap between BPML and WSCI. See [11] for more information about the evaluation of BPML and WSCI using the patterns.

In addition to comparing XPDL to other standards, it is interesting to compare XPDL with contemporary WFM systems. Tables 2 and 3 summarize the results of the comparison of 15 WFM systems in terms of the selected patterns. These tables are taken from [15] and have been added to compare contemporary workflow products with XPDL.

| pattern | product | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Staffware | COSA | InConcert | Eastman | FLOWer | Domino | Meteor | Mobile |
| 1 (seq) | + | + | + | + | + | + | + | + |
| 2 (par-spl) | + | + | + | + | + | + | + | + |
| 3 (synch) | + | + | + | + | + | + | + | + |
| 4 (ex-ch) | + | + | +/- | + | + | + | + | + |
| 5 (simple-m) | + | + | +/- | + | + | + | + | + |
| 6 (m-choice) | - | + | +/- | +/- | - | + | + | + |
| 7 (sync-m) | - | +/- | + | + | - | + | - | - |
| 8 (multi-m) | - | - | - | + | +/- | +/- | + | - |
| 9 (disc) | - | - | - | + | +/- | - | +/- | + |
| 10 (arb-c) | + | + | - | + | - | + | + | - |
| 11 (impl-t) | + | - | + | + | - | + | - | - |
| 12 (mi-no-s) | - | +/- | - | + | + | +/- | + | - |
| 13 (mi-dt) | + | + | + | + | + | + | + | + |
| 14 (mi-rt) | - | - | - | - | + | - | - | - |
| 15 (mi-no) | - | - | - | - | + | - | - | - |
| 16 (def-c) | - | + | - | - | +/- | - | - | - |
| 17 (int-par) | - | + | - | - | +/- | - | - | + |
| 18 (milest) | - | + | - | - | +/- | - | - | - |
| 19 (can-a) | + | + | - | - | +/- | - | - | - |
| 20 (can-c) | - | - | - | - | +/- | + | - | - |

**Table 2.** The main results for Staffware, COSA, InConcert, Eastman, FLOWer, Lotus Domino Workflow, Meteor, and Mobile.

From the comparison it is clear that no tool supports all of the selected patterns. In fact, many of these tools only support a relatively small subset of the more advanced patterns (i.e., patterns 6 to 20). Specifically the limited support for the discriminator, the state-based patterns (only COSA), the synchronization of multiple instances (only FLOWer) and cancellation (esp. of activities), is worth noting.

Please apply the results summarized in tables 1, 2 and 3 with care. First of all, the organization selecting a WFM system/standard should focus on the patterns most relevant for the workflow processes at hand. Since support for the more advanced patterns is limited, one should focus on the patterns most needed. Second, the fact that a pattern is not directly supported by a product does not imply that it is not possible to support

| pattern | product | | | | | | |
|---|---|---|---|---|---|---|---|
| | MQSeries | Forté | Verve | Vis. WF | Changeng. | I-Flow | SAP/R3 |
| 1 (seq) | + | + | + | + | + | + | + |
| 2 (par-spl) | + | + | + | + | + | + | + |
| 3 (synch) | + | + | + | + | + | + | + |
| 4 (ex-ch) | + | + | + | + | + | + | + |
| 5 (simple-m) | + | + | + | + | + | + | + |
| 6 (m-choice) | + | + | + | + | + | + | + |
| 7 (sync-m) | + | - | - | - | - | - | - |
| 8 (multi-m) | - | + | + | - | - | - | - |
| 9 (disc) | - | + | + | - | + | - | + |
| 10 (arb-c) | - | + | + | +/- | + | + | - |
| 11 (impl-t) | + | - | - | - | - | - | - |
| 12 (mi-no-s) | - | + | + | + | - | + | - |
| 13 (mi-dt) | + | + | + | + | + | + | + |
| 14 (mi-rt) | - | - | - | - | - | - | +/- |
| 15 (mi-no) | - | - | - | - | - | - | - |
| 16 (def-c) | - | - | - | - | - | - | - |
| 17 (int-par) | - | - | - | - | - | - | - |
| 18 (milest) | - | - | - | - | - | - | - |
| 19 (can-a) | - | - | - | - | - | - | + |
| 20 (can-c) | - | + | + | - | + | - | + |

**Table 3.** The main results for MQSeries, Forté Conductor, Verve, Visual WorkFlo, Changengine, I-Flow, and SAP/R3 Workflow.

the construct at all. As indicated in [15], many patterns can be supported indirectly through mixtures of more basic patterns and coding. Third, the patterns reported in this tutorial only focus on the process perspective (i.e., control flow or routing). The other perspectives (e.g., organizational modeling) should also be taken into account.

Tables 1, 2 and 3 allow for an objective comparison of the 7 standards and 15 WFM systems. When comparing XPDL to the 6 other standards, it is remarkable to see that XPDL seems to be less expressive than web service composition languages such as BPEL4WS and BPML. An important pattern like the Deferred choice (WP16) is supported by most standards and is vital for practical application of WFM. Nevertheless, it is not even mentioned in [75]. Compared to the 15 WFM systems, XPDL is not as expressive as one would expect. Many systems offer functionality (e.g., the Deferred choice and the Cancel activity patterns), not supported by XPDL. It almost seems that XPDL offers the intersection rather than the union of the functionality offered by contemporary systems. This may have been the initial goal of XPDL. However, if this is the case, two important questions need to be answered.

1. If XPDL offers the intersection rather than the union of the functionality of existing systems, then how to use XPDL in practice? Should workflow designers that want to be able to export only use a subset of the functionality offered by the system? If so, users would not be able to use powerful concepts like the Deferred choice (WP16) and the Cancel activity (WP19) patterns.

2. Why does XPDL support the Synchronizing merge (WP7) while it is only supported by a few systems. Widely-used systems like Staffware do not support this pattern, and therefore, will be unable to interpret the AND-join as indicated in [75].

Note that the issues raised cannot be solved satisfactorily. If XPDL offers the intersection of the functionality of existing systems, it is less expressive than many of the existing tools and standards. If XPDL offers the union of available functionality, it may become impossible to import a process definition into a concrete system and interpret it correctly. (Recall that no system supports all patterns.) Unfortunately, this dilemma is not really addressed by the WfMC [75]. The introduction of extended attributes (i.e., extensions of XPDL for a specific product) and conformance classes (i.e., restrictions to allow the use of specific products) are no solution and only complicate matters.

There have been several comparisons of some of the languages mentioned in this tutorial. These comparisons typically do not use a framework and provide an opinion rather than a structured analysis. A positive example is [65] where XPDL, BPML and BPEL4WS are compared by relating the concepts used in the three languages. Unfortunately, the paper raises more questions than it answers.

Besides the dilemma that XPDL is either not expressive enough or too expressive, there is the problem of semantics. In [75] the WfMC does not give unambiguous specification of all the elements in the language. As a result, many vendors can claim to be compliant while interpreting constructs in a different way. In Section 5.4, we demonstrated that there are many interpretations of seemingly basic constructs like the AND-join and XOR-join. The lack of semantics restricts the application of XPDL and does not allow for a meaningful realization of the topic of conformance. As indicated before, [75] defines conformance as follows: “A product that claims conformance must generate valid, syntactically correct XPDL, and must be able to read all valid XPDL.”. Clearly, this inadequate and will not stimulate further standardization in the workflow domain. As a result, web service composition languages like BPML and BPEL4WS may take over the role of XPDL [6].

## 6 Related work

There is a lot of literature on WFM and WFM systems. Only some of the books on WFM are referred to in this tutorial [12, 37, 38, 48, 55, 57, 58, 61]. There are also many publications reporting on the application of Petri nets to WFM. In this tutorial we mainly referred to papers using WF-nets and soundness (or variants thereof) [1, 3, 4, 28, 44, 52, 54]. For the evaluation of XPDL we relied heavily on the work on workflow patterns. See [15, 77] or `http://www.workflowpatterns.com` for more information. The two Springer Lecture Notes in Computing science volumes on BPM can serve as a starting point for finding the state-of-the-art results in this domain [10, 16]. Clearly, it is impossible to be complete. Please use the references given in this tutorial for finding more material. Finally, we would like to point out `http://www.workflowcourse.com` as a resource for all kinds of learning material ranging from slides to interactive animations.

## 7 Conclusion

The goal of this tutorial is to introduce the WFM/BPM domain from a Petri-net point-of-view. The focus of the first part of the tutorial was on the application of Petri nets in this domain. Sections 2 and 3 showed how WF-nets can be used to model and analyze workflow processes. The focus of the second part was more on the application domain itself. Sections 4 and 5 provided information on systems, languages, and standards. To illustrate things we presented a detailed analysis of XPDL using a set of workflow patterns.

To conclude this tutorial we reflect on the role of Petri nets in the WFM/BPM domain. There are at least three good reasons for using Petri nets for workflow modeling and analysis ([2]):

1. *Formal semantics despite the graphical nature*
   On the one hand, Petri nets are a graphical language which allows for the modeling of the workflow primitives identified by the WfMC. On the other hand, the semantics of Petri nets (including most of the extensions) have been defined formally. Many of today's available WFM systems provide ad-hoc constructs to model workflow procedures. Moreover, there are WFM systems that impose restrictions on many of the workflow patterns discussed. Some WFM systems also provide exotic constructs whose semantics are not 100% clear, cf. the join construct in XPDL and many other languages. Because of these problems it is better to use a well-established design language with formal semantics as a solid basis.
2. *State-based instead of event-based*
   In contrast to many other process modeling techniques, the state of case can be modeled explicitly in a Petri net. Process modeling techniques ranging from informal techniques such as dataflow diagrams to formal techniques such as process algebra's are *event-based*, i.e., transitions are modeled explicitly and the states between subsequent transitions are only modeled implicitly. Today's WFM systems are typically event-based, i.e., tasks are modeled explicitly and states between subsequent tasks are suppressed. The distinction between an event-based and a state-based description may appear to be subtle, but patterns like the Deferred choice (WP16) and the Milestone (WP18) show that this is of the utmost importance for workflow modeling.
3. *Abundance of analysis techniques*
   Petri nets are marked by the availability of many analysis techniques. Clearly, this is a great asset in favor of a Petri nets. In this tutorial, we focused on the verification of WF-nets. We have seen that Petri-net-based analysis techniques can be used to determine the correctness of a workflow process definition. The availability of these techniques illustrates that Petri-net theory can be used to add powerful analysis capabilities to the next generation of WFM systems.

However, as indicated in [13] there are also problems when modeling workflows in terms of a Petri nets. For the more advanced routing constructs it is necessary to resort to high-level nets [49, 50]. Moreover, a straightforward application of high-level Petri nets does not always yield the desired result. There seem to be three problems relevant for modeling workflow processes:

1. High-level Petri nets support colored tokens, i.e., a token can have a value. Although it is possible to use this to identify multiple instances of a subprocess, there is no specific support for *patterns involving multiple instances* and the burden of keeping track, splitting, and joining of instances is carried by the designer.
2. Sometimes two flows need to be joined while it is not clear whether synchronization is needed, i.e., if both flows are active an AND-join is needed otherwise an XOR-join. Such *advanced synchronization patterns* are difficult to model in terms of a high-level Petri net because the firing rule only supports two types of joins: the AND-join (transition) or the XOR-join (place).
3. The firing of a transition is always local, i.e., enabling is only based on the tokens in the input places and firing is only affecting the input and output places. However, some events in the workflow may have an effect which is not local, e.g., because of an error tokens need to be removed from various places without knowing where the tokens reside. Everyone who has modeled such a *cancellation pattern* (e.g., a global timeout mechanism) in terms of Petri nets knows that it is cumbersome to model a so-called "vacuum cleaner" removing tokens from selected parts of the net.

Compared to existing WFM languages high-level Petri nets are quite expressive when it comes to supporting the workflow patterns. Recall that we use the term "expressiveness" not in the formal sense. High-level Petri nets are Turing complete, and therefore, can do anything we can define in terms of an algorithm. However, this does not imply that the modeling effort is acceptable. High-level nets, in contrast to many workflow languages, have no problems dealing with state-based patterns. This is a direct consequence of the fact that Petri nets use places to represent states explicitly. Although high-level Petri nets outperform most of the existing languages when it comes to modeling the control flow, the result is not completely satisfactory since the three problems indicated hamper the application in the WFM/BPM domain. This triggered the development of *YAWL (Yet Another Workflow Language)*. YAWL is based on Petri nets but extended with additional features to facilitate the modeling of complex workflows [13, 14]. See `http://www.citi.qut.edu.au/yawl/` for more information or to download the YAWL system.

**Acknowledgments.** The author would like to thank Lachlan Aldred, Alistair Barros, Twan Basten, Marlon Dumas, Bartek Kiepuszewski, Kees van Hee, Akhil Kumar, Arthur ter Hofstede, Hajo Reijers, Eric Verbeek, Ton Weijters, and Petia Wohed for their collaborative work on the topics discussed in this tutorial.

**Disclaimer.** The author and the associated institutions assume no legal liability or responsibility for the accuracy and completeness of any information about XPDL or any of the other standards/products mentioned in this tutorial. However, all possible efforts have been made to ensure that the results presented are, to the best of our knowledge, up-to-date and correct.

## References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.

2. W.M.P. van der Aalst. Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System. In T. Wakayama, S. Kannapan, C.M. Khoong, S. Navathe, and J. Yates, editors, *Information and Process Integration in Enterprises: Rethinking Documents*, volume 428 of *The Kluwer International Series in Engineering and Computer Science*, pages 161–182. Kluwer Academic Publishers, Boston, Massachusetts, 1998.
3. 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.
4. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
5. W.M.P. van der Aalst. Making Work Flow: On the Application of Petri nets to Business Process Management. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Berlin, 2002.
6. W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
7. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
8. W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51. ACM Press, New York, 2001.
9. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.
10. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.
11. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-Based Analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.
12. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
13. W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, volume 560 of *DAIMI*, pages 1–20, Aarhus, Denmark, August 2002. University of Aarhus.
14. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.
15. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
16. W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors. *Busines Process Management*, volume 2678 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.
17. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

18. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.
19. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
20. A. Arkin, S. Askary, S. Fordin, and W. Jekel et al. Web Service Choreography Interface (WSCI) 1.0. Standards propsal by BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.
21. A. Arkin et al. Business Process Modeling Language (BPML), Version 1.0, 2002.
22. Pallas Athena. *Case Handling with FLOWer: Beyond workflow*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
23. K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, Berlin, 1995.
24. J. Billington and et. al. The Petri Net Markup Language: Concepts, Technology, and Tools. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–506. Springer-Verlag, Berlin, 2003.
25. R. Casonato. Gartner Group Research Note 00057684, Production-Class Workflow: A View of the Market. http://www.gartner.com, 1998.
26. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In R.K. Shyamasundar, editor, *Foundations of software technology and theoretical computer science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, 1993.
27. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0. Standards propsal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
28. J. Dehnert. *A Methodology for Workflow Modeling: From Business Process Modeling Towards Sound Workflow Specification*. PhD thesis, TU Berlin, Berlin, Germany, 2003.
29. J. Desel. A proof of the Rank theorem for extended free-choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, Berlin, 1992.
30. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
31. M. Dumas and A.H.M. ter Hofstede. UML activity diagrams as a workflow specification language. In M. Gogolla and C. Kobryn, editors, *Proc. of the 4th Int. Conference on the Unified Modeling Language (UML01)*, volume 2185 of *LNCS*, pages 76–90, Toronto, Canada, October 2001. Springer Verlag.
32. 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.
33. C.A. Ellis and G. Nutt. Workflow: The Process Spectrum. In A. Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, Athens, Georgia, May 1996.
34. R. Eshuis and J. Dehnert. Reactive Petri nets for Workflow Modeling. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 295–314. Springer-Verlag, Berlin, 2003.
35. J. Esparza. Synthesis rules for Petri nets, and how they can lead to new results. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198. Springer-Verlag, Berlin, 1990.

36. J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.
37. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.
38. L. Fischer, editor. *Workflow Handbook 2003, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2003.
39. Gartner. Gartner's Application Development and Maintenance Research Note M-16-8153, The BPA Market Cathes another Major Updraft. http://www.gartner.com, 2002.
40. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
41. R.J. van Glabbeek and D.G. Stork. Query Nets: Interacting Workflow Modules that Ensure Global Termination. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 184–199. Springer-Verlag, Berlin, 2003.
42. Object Management Group. *OMG Unified Modeling Language 2.0 Proposal, Revised submission to OMG RFPs ad/00-09-01 and ad/00-09-02, Version 0.671*. OMG, http://www.omg.com/uml/, 2002.
43. M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.
44. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
45. A. W. Holt. Coordination Technology and Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, Berlin, 1985.
46. IBM. *IBM MQSeries Workflow - Getting Started With Buildtime*. IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999.
47. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.
48. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
49. K. Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer-Verlag, Berlin, 1990.
50. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
51. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.tm.tue.nl/it/research/patterns.
52. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.
53. E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, June 1999.
54. E. Kindler, A. Martens, and W. Reisig. Inter-Operability of Workflow Applications: Local Criteria for Global Soundness. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer-Verlag, Berlin, 2000.

55. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.
56. F. Leymann. Web Services Flow Language, Version 1.0, 2001.
57. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
58. D.C. Marinescu. *Internet-Based Workflow Management: Towards a Semantic Web*, volume 40 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, New York, 2002.
59. M. Ajmone Marsan, G. Balbo, and G. Conte et al. *Modelling with Generalized Stochastic Petri Nets*. Wiley series in parallel computing. Wiley, New York, 1995.
60. A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.
61. M. Zur Muehlen. *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems*. Logos, Berlin, 2004.
62. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer-Verlag, Berlin, 1985.
63. 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.
64. W. Sadiq and M.E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.
65. R. Shapiro. A Comparison of XPDL, BPML and BPEL4WS (Version 1.4). http://xml.coverpages.org/Shapiro-XPDL.pdf, 2002.
66. Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1998.
67. Software-Ley. *COSA 3.0 User Manual*. Software-Ley GmbH, Pullheim, Germany, 1999.
68. S. Staab, W. van der Aalst, V.R. Benjamins, A. Sheth, J. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. Web Services: Been There, Done That? (Trends and Controversies). *IEEE Intelligent Systems*, 18(1):72–85, 2003.
69. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 2000.
70. S. Thatte. XLANG Web Services for Business Process Design, 2001.
71. R. van der Toorn. *Component-Based Software Design with Petri nets: An Approach Based on Inheritance of Behavior*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.
72. H.M.W. Verbeek. *Verification and Enactment of Workflow Management Systems (submitted)*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.
73. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
74. WFMC. Workflow Management Coalition Workflow Standard: Interface 1 – Process Definition Interchange Process Model (WFMC-TC-1016). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 1999.
75. WFMC. Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface – XML Process Definition Language (XPDL) (WFMC-TC-1025). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.
76. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
77. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.

78. M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.

## A XPDL Schema

The listing below shows selected parts of the XPDL Schema given in [75] relevant for this tutorial.

```
<xsd:element name="Activity">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Description" minOccurs="0"/>
         <xsd:element ref="xpdl:Limit" minOccurs="0"/>
         <xsd:choice>
            <xsd:element ref="xpdl:Route"/>
            <xsd:element ref="xpdl:Implementation"/>
            <xsd:element ref="xpdl:BlockActivity"/>
         </xsd:choice>
         <xsd:element ref="xpdl:Performer" minOccurs="0"/>
         <xsd:element ref="xpdl:StartMode" minOccurs="0"/>
         <xsd:element ref="xpdl:FinishMode" minOccurs="0"/>
         <xsd:element ref="xpdl:Priority" minOccurs="0"/>
         <xsd:element ref="xpdl:Deadline" minOccurs="0"
                      maxOccurs="unbounded"/>
         <xsd:element ref="xpdl:SimulationInformation" minOccurs="0"/>
         <xsd:element ref="xpdl:Icon" minOccurs="0"/>
         <xsd:element ref="xpdl:Documentation" minOccurs="0"/>
         <xsd:element ref="xpdl:TransitionRestrictions" minOccurs="0"/>
         <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="Name" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
...
<xsd:element name="TransitionRestriction">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Join" minOccurs="0"/>
         <xsd:element ref="xpdl:Split" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element> <xsd:element name="TransitionRestrictions">
   <xsd:complexType>
      <xsd:sequence>
```

```
         <xsd:element ref="xpdl:TransitionRestriction" minOccurs="0"
                      maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
...
<xsd:element name="Join">
   <xsd:complexType>
      <xsd:attribute name="Type">
         <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
               <xsd:enumeration value="AND"/>
               <xsd:enumeration value="XOR"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
   </xsd:complexType>
</xsd:element>
...
<xsd:element name="Split">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:TransitionRefs" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Type">
         <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
               <xsd:enumeration value="AND"/>
               <xsd:enumeration value="XOR"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
   </xsd:complexType>
</xsd:element>
...
<xsd:element name="Transition">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Condition" minOccurs="0"/>
         <xsd:element ref="xpdl:Description" minOccurs="0"/>
         <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="From" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="To" type="xsd:NMTOKEN" use="required"/>
```

```
      <xsd:attribute name="Name" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
...
<xsd:element name="Condition">
   <xsd:complexType mixed="true">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="xpdl:Xpression"/>
      </xsd:choice>
      <xsd:attribute name="Type">
         <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
               <xsd:enumeration value="CONDITION"/>
               <xsd:enumeration value="OTHERWISE"/>
               <xsd:enumeration value="EXCEPTION"/>
               <xsd:enumeration value="DEFAULTEXCEPTION"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
   </xsd:complexType>
</xsd:element>
```