# Beautiful Workflows: A Matter of Taste?

Wil M.P. van der Aalst[1,2,3], Michael Westergaard[1,2], and Hajo A. Reijers[1,4]

[1] Architecture of Information Systems, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
`{w.m.p.v.d.aalst,m.westergaard,h.a.reijers}@tue.nl`
[2] International Laboratory of Process-Aware Information Systems, National Research University Higher School of Economics (HSE),
33 Kirpichnaya Str., Moscow, Russia.
[3] Business Process Management Discipline, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia.
[4] Perceptive Software,
Piet Joubertstraat 4, 7315 AV Apeldoorn, The Netherlands.

**Abstract.** Workflows can be specified using different languages. Mainstream workflow management systems predominantly use procedural languages having a graphical representation involving AND/XOR splits and joins (e.g., using BPMN). However, there are interesting alternative approaches. For example, case handling approaches are data-driven and allow users to deviate within limits, and declarative languages based on temporal logic (where everything is allowed unless explicitly forbidden). Recently, Rinus Plasmeijer proposed the iTask system (iTasks) based on the viewpoint that workflow modeling is in essence a particular kind of functional programming. This provides advantages in terms of expressiveness, extendibility, and implementation efficiency. On the occasion of his 61st birthday, we compare four workflow paradigms: *procedural*, *case handling*, *declarative*, and *functional*. For each paradigm we selected a characteristic workflow management system: YAWL (procedural), BPM|one (case handling), Declare (declarative), and iTasks (functional). Each of these systems aims to describe and support business processes in an elegant manner. However, there are significant differences. In this paper, we aim to identify and discuss these differences.

**Keywords:** Workflow Management, Business Process Management, Case Handling, Declarative Languages, Functional Programming

## 1 Demand Driven Workflow Systems

Functional programming and process modeling are related in various ways. For example, well-known Petri nets tools such as CPN Tools [14] and ExSpect [6] use functional languages to describe the consumption and production behaviors of transitions in the Petri net. However, the different communities focusing on process modeling and analysis are largely disconnected from the functional programming community (and vice versa). Business Process Management (BPM), Workflow Management (WFM), and concurrency-related (e.g., Petri nets) communities are rarely using concepts originating from functional languages. Therefore, the groups of Rinus Plasmeijer and Wil van der Aalst submitted the joint project proposal "Controlling Dynamic Real Life Workflow Situations with Demand Driven Workflow Systems" to STW in 2006. The project was accepted in 2007 and started in 2008. The project completed successfully in 2012.

In the STW project different styles of workflow modeling and enactment were used. A new style of functional programming, called *Task-Oriented Programming* (TOP), was developed by Rinus and his team [39]. The *iTask* system (iTasks), an implementation of TOP embedded in the well-known functional language Clean, supports this style of workflow development [37, 39]. iTasks workflows consist of typed tasks that produce results that can be passed as parameters to other tasks. New combinators can be added to extend the iTasks language. At Eindhoven University of Technology, Maja Pesic and Michael Westergaard worked on an alternative approach based on the Declare system. The Declare language is based on the notion of constraints, grounded in LTL, and also extendible. Moreover, previously we worked on procedural workflow languages like YAWL and collaborated with Pallas Athena on the case handling paradigm.[5]

In the project we could experience the enthusiasm, dedication, an persistence of Rinus when it comes to functional programming and beautiful code. Therefore, it is an honor to be able to contribute to this festschrift devoted to the 61st birthday of Rinus Plasmeijer!

In the remainder, we report on insights obtained in our joint STW project. In Section 2 we discuss four different workflow paradigms using four representative examples: YAWL (procedural), BPM|one (case handling), Declare (declarative), and iTasks (functional). Section 3 compares the different paradigms and reflects on the current BPM/WFM market. Section 4 concludes our contribution to this festschrift.

## 2 Four Workflow Paradigms

Business Process Management (BPM) is the discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes [1, 3, 46]. It has received considerable attention in recent years due to its potential for significantly increasing productivity and saving costs. Moreover, today there is an abundance of BPM systems. These systems are *generic software systems that are driven by explicit process designs to enact and manage operational business processes* [3].

BPM can be seen as an extension of *Workflow Management* (WFM). WFM primarily focuses on the automation of business processes [8, 28, 30], whereas BPM has a broader scope: from process automation and process analysis to operations management and the organization of work. BPM aims to improve

[5] Pallas Athena was also involved in the User Committee of our joint STW project.

operational business processes, with or without the use of new technologies. For example, by modeling a business process and analyzing it using simulation, management may get ideas on how to reduce costs while improving service levels. Moreover, BPM is often associated with software to manage, control, and support operational processes. This was the initial focus of WFM. However, traditional WFM technology aimed at the automation of business processes in a rather mechanistic manner without much attention for human factors and management support.

In the remainder we use the terms WFM and BPM interchangeably as we focus on the modeling and enactment of business processes, i.e., the emphasis will be on process automation rather than management support.

We identify four very different styles of process automation: procedural workflows (YAWL), case handling workflows (BPM|ONE), declarative workflows (DECLARE), and functional workflows (iTASKS). In the remainder of this section, these are introduced and subsequently compared in Section 3.

## 2.1 Procedural Workflows (YAWL)

Procedural programming (also referred to as imperative programming) aims to define sequences of commands for the computer to perform in order to reach a predefined goal. Procedural programming can be seen as the opposite of more declarative forms of programming that define *what* the program should accomplish without prescribing *how* to do it in terms of sequences of actions to be taken. Despite criticism, procedural programming is still the mainstream programming paradigm. A similar observation can be made when looking at the modeling, analysis, and enactment of business processes. Almost all BPM/WFM languages and tools are procedural (see also Section 3.4). Examples are BPMN (Business Process Modeling Notation), UML activity diagrams, Petri nets, process calculi like CSP and CCS, BPEL (Business Process Execution Language), and EPCs (Event-driven Process Chains).

Figure 1(a) describes a simple diagnosis process in terms of Petri, a *WF-net* (WorkFlow net) to be precise [13, 26, 46]. Tasks are modeled by labeled transitions and the ordering of these tasks is controlled by places (represented by circles). A transition (represented by a square) is enabled if each of its input places contains a token. An enabled transition may occur thereby consuming a token from each input place and producing a token for each output place. The process in Figure 1(a) starts with a token in place *in* (depicted by a black dot). Transition *a* (*admit patient*) can occur if there is a token in place *in*. Firing *a* corresponds to removing the token from place *in* and producing a token for place $p1$. After admitting the patient (modeled by transition *a*), vital signs may be checked (*b*) or not (modeled by the silent transition). Then the physical examination (*c*) is conducted. Subsequently, the blood is tested (*d*), an X-ray is taken (*e*), and an ECG is made (*f*) (any ordering is allowed). In the last step, the diagnosis is finalized (*g*). The process instance terminates when place *out* is marked. Figure 1(c) shows an event log describing some example traces of the

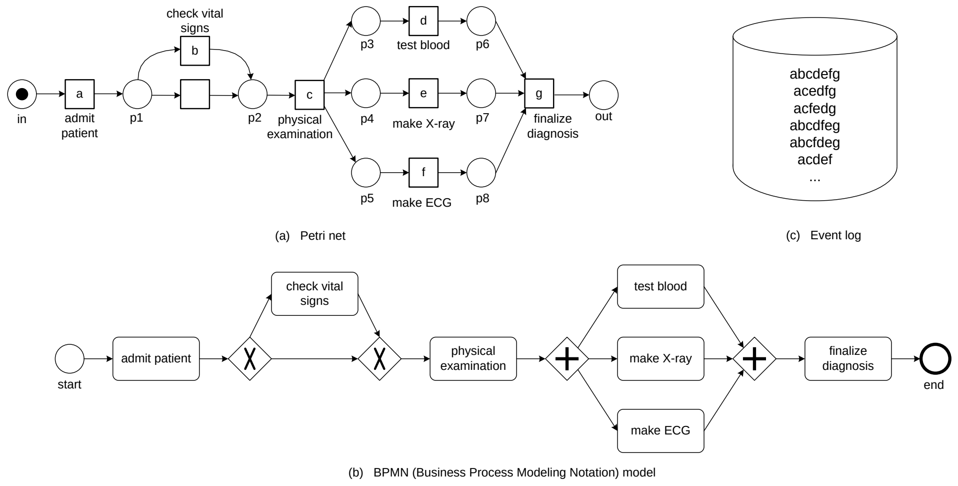

**Fig. 1.** A Petri net (a) and BPMN model (b) describing a simple medical diagnosis process. Example traces are shown in the event log (c), $a$ = *admit patient*, etc.

model. The WF-net allows for 12 different executions: $d$, $e$ and $f$ can be executed in any order and $b$ may be skipped.

BPMN, EPCs, UML ADs, and many other business process modeling notations have in common that they all use *token-based semantics* [3, 13, 21, 26, 46]. Therefore, there are many techniques and tools to convert Petri nets to BPMN, BPEL, EPCs and UML ADs, and vice versa. As a result, the core concepts of Petri nets are often used indirectly, e.g., to enable analysis, to enact models, and to clarify semantics. For example, Figure 1(b) shows the same control-flow modeled using the *Business Process Modeling Notation* (BPMN) [34]. BPMN uses activities, events, and gateways to model the control-flow. In Figure 1(b) two types of gateways are employed: exclusive gateways are used to model XOR-splits and joins and parallel gateways are used to model AND-splits and joins. BPMN also supports other types of gateways corresponding to inclusive OR-splits and joins, deferred choices, etc. [21, 26, 46].

In an effort to gain a better understanding of the fundamental concepts underpinning business processes, the *Workflow Patterns Initiative*[6] was conceived in the late nineties with the goal of identifying the core architectural constructs inherent in workflow technology [10]. The original objective was to delineate the fundamental requirements that arise during business process modeling on a recurring basis and describe them in an imperative way. The main driver for the Workflow Patterns Initiative was the observation that WFM/BPM languages and tools differed markedly in their expressive power and the range of concepts that they were able to capture. The initial set of 20 patterns provided a basis for valuable comparative discussions on the capabilities of languages and systems.

[6] See `www.workflowpatterns.com`.

Later the original set of 20 patterns was extended into a set of 43 control-flow patterns supported by additional sets of patterns, e.g., 40 data patterns and 43 resource patterns [26].

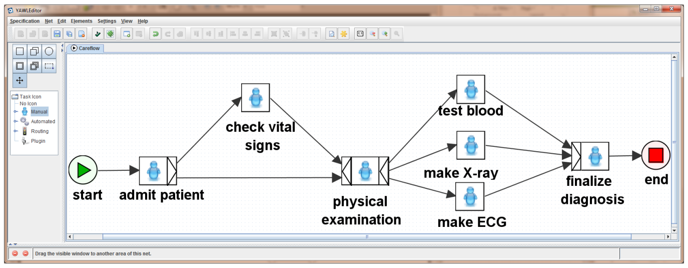

**Fig. 2.** Screenshot of YAWL editor while modeling the process described using Figure 1.

The workflow patterns provided the conceptual basis for the YAWL language [9] and YAWL workflow system [26].[7] YAWL supports most workflow patterns directly, i.e., no workarounds are needed to model and support a wide variety of imperative process behaviors. Petri nets were taken as a starting point for YAWL and extended with dedicated constructs to deal with patterns that Petri nets have difficulty expressing, in particular patterns dealing with cancelation, synchronization of active branches only, and multiple concurrently executing instances of the same task. The screenshot in Figure 2 shows the YAWL variant of the diagnosis process introduced using earlier. The YAWL model allows for the same 12 traces as allowed by the WF-net and BPMN model in Figure 1. None of the advanced features of YAWL are needed to model this simple diagnosis process. Note that compared to the Petri net there are no explicit places. In YAWL one can connect two tasks directly without inserting a place. However, internally the places are added to model states. Moreover, for workflow patterns such as the *deferred choice* pattern (the decision is not made automatically from within the context of the process but is deferred to an entity in the operational environment) and the *milestone* pattern (the additional restriction that a task can only proceed when another concurrent branch of the process has reached a specific state), places need to be represented explicitly to model the desired behavior. The aim of YAWL is to offer direct support for many patterns while keeping the language simple. It can be seen as a reference implementation of the most important workflow patterns. Over time, the YAWL language and the YAWL system have increasingly become synonymous and have garnered widespread interest from both practitioners and the academic community alike. Over time YAWL evolved into one of the most widely used open-source workflow systems.

[7] YAWL can be downloaded from `www.yawlfoundation.org`.

Most mainstream WFM/BPM languages are procedural and use a token-based semantics. The same holds for *analysis techniques* relevant for WFM/BPM efforts. Most model-based analysis techniques ranging from verification to performance analysis are tailored towards procedural models. *Verification* is concerned with the correctness of a system or process. *Performance analysis* focuses on flow times, waiting times, utilization, and service levels. Also *process mining* techniques driven by event data typically assume procedural models. For example, *process discovery* techniques can be used to learn procedural models based on event data. *Conformance checking* techniques compare procedural models (modeled behavior) with event data (observed behavior).

There is an abundance of analysis techniques developed for Petri nets ranging from verification and simulation [13] to process mining [2]. Procedural languages such as BPMN, UML activity diagrams, BPEL, and EPCs can be converted to Petri nets for verification, performance analysis, and conformance checking. Petri nets can be mapped onto mainstream notations to visualize processes discovered using process mining.

### 2.2 Case Handling Workflows (BPM|one)

Mainstream procedural languages are often criticized for being inflexible. *Case handling* is a paradigm for supporting *flexible* and *knowledge intensive* business processes [15]. It is strongly based on data as the typical product of these processes. Unlike traditional WFM systems, which use predefined process control structures to determine what should be done during a workflow process, case handling focuses on what can be done to achieve a business goal. In case handling, the knowledge worker in charge of a particular case actively decides on how the goal of that case is reached, and the role of a case handling system is assisting rather than guiding her in doing so. The core features of case handling are:

- *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 tasks are *enabled on the basis of the information available* rather than the tasks already executed,
- *allow for deviations* (without certain bounds) that are not explicitly modeled (skip, redo, etc.),
- *separate* work distribution from authorization,
- allow workers to *view and add/modify data* before or after the corresponding tasks have been executed (e.g., information can be registered the moment it becomes available).

The central concept for case handling is the *case* and not the tasks or the ordering of tasks. The case is the “product” which is manufactured, and at any time workers should be aware of this context. For knowledge-intensive processes, *the state and structure of a case can be derived from the relevant data objects.* A data object is a piece of information which is present or not present and when it

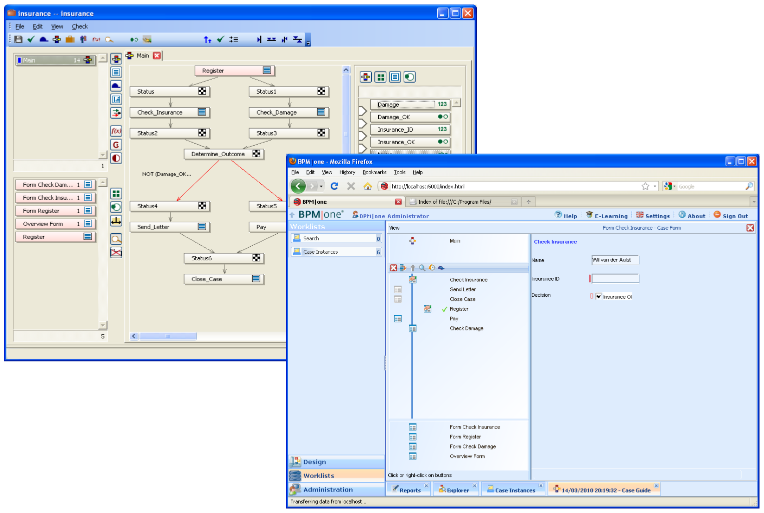

**Fig. 3.** Screenshot showing the BPM|ONE designer and worklist handler.

is present it has a value. In contrast to existing workflow management systems, the state of the case is not determined by the control-flow status but by the presence of data objects. This is truly a paradigm shift: case handling is also driven by data-flow and not just by control-flow.

In a procedural workflow, workers need to execute all tasks offered by the system and there is no way to later correct errors if not modeled explicitly. Case handling allows for deviations within certain bounds. For a task at least three types of roles can be specified:

– The *execute* role is the role that is necessary to carry out the task or to start a process.
– The *redo* role is necessary to undo tasks, i.e., the case returns to the state before executing the task. Note that it is only possible to undo a task if all following tasks are undone as well.
– The *skip* role is necessary to bypass tasks, e.g., a check may be skipped by a manager but not by a regular employee.

Case handling is supported by only a few vendors. The best-known example is BPM|ONE which is now part of the Perceptive Platform (Lexmark).[8] BPM|ONE is the successor of FLOWER [15] both developed by Pallas Athena. In turn, FLOWER was inspired by the ECHO (Electronic Case-Handling for Offices) system whose development started in 1986 within Philips and later moved to Digital. BPM|ONE supports all of the concepts mentioned (see Figure 3). The system is much more flexible than most WFM/BPM systems. This is achieved without forcing end-users to adapt process models (which is typically infeasible).

[8] See `http://www.perceptivesoftware.com/products/perceptive-process/business-process-management`.

Consider the process described in Figure 1. Using BPM|ONE one could make some tasks “skipable”, e.g., task *make ECG* may be skipped by the department chief even though this is not modeled. Similarly, some tasks may be “redoable”, e.g., after doing the blood test, the process may be rolled-back to the task *physical examination.* Moreover, tasks may be data driven. If a recent X-ray is available, task *make X-ray* is completed automatically without actually making a new X-ray.

Recently, the terms Adaptive Case Management (ACM) and Dynamic Case Management (DCM) received quite some attention. These terms are used to stress the need for workflows to be more human-centric, flexible, and content and collaboration driven. Unfortunately, the different vendors interpret these terms in different ways and the actual case-handling functionality described before is often missing.

### 2.3 Declarative Workflows (Declare)

The procedural paradigm focuses on *how* to accomplish a goal. At any point a user is presented with a relatively limited selection of possible tasks based on explicit decisions. Declarative workflows instead focus on the *relationship* between tasks, such as one task cannot be executed together with another or one task has to be followed by another [12]. For example, in a medical treatment, one type of medicine may be incompatible with another and cannot be used together, or surgery must be followed up by retraining.

Declarative workflows therefore focus on two things, *tasks* to be executed and *constraints* between the tasks, stating properties the aforementioned [12]. In Fig. 4, we see an example of a simple medical process for treating wounds modeled using DECLARE.[9] Tasks are represented as rectangles, e.g., Disinfect Wound and constraints are either represented as arcs between tasks (for binary constraints) or as annotations of tasks (for unary constraints). For example, we have a init constraint on the Receive Patient task, and a precedence constraint from Disinfect Wound to Stitch Wound. We also have a response constraint from Stitch Wound to Prescribe Antibiotics, a non co-existence constraint between Prescribe Antibiotics and Give Tetanus Vaccine, and a second precedence constraint from Disinfect Wound to Give Tetanus Vaccine.

Constraints have informal descriptive semantics and formal semantics. The informal semantics suffice for users and the formal semantics are only used by implementers and for reasoning about processes. We have summed up the formal semantics of four example DECLARE constraints in Table 1. DECLARE comes with many more constraints, but these suffice here for understanding the basic idea. We first consider the informal semantics for these constraints. The init constraint models that any execution has to start with this task, and in our example this means that the first thing to do in any execution is to receive a patient. The precedence constraint models that it is not legal to execute the target before executing the source of the constraint. In the example, we model that we may not

[9] DECLARE can be downloaded from `www.win.tue.nl/declare/`.

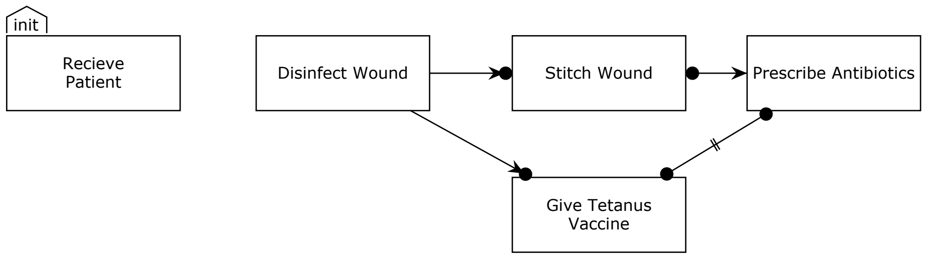

**Fig. 4.** DECLARE model of simple medical treatment.

**Table 1.** Selected DECLARE constraints.

| Name | Parameters | Representation | Semantics | |
|---|---|---|---|---|
| | | | Informal | Formal |
| init | $(A)$ | $\overset{\widehat{init}}{\boxed{A}}$ | start with A | $\bar{\mathbf{X}}A$ |
| precedence | $(A, B)$ | $\boxed{A}\longrightarrow\!\bullet\boxed{B}$ | no B before A | $A\mathbf{R}\neg B$ |
| response | $(A, B)$ | $\boxed{A}\bullet\!\!\longrightarrow\boxed{B}$ | after A must execute B | $\mathbf{G}(A \rightarrow \mathbf{F}B)$ |
| not no-existence | $(A, B)$ | $\boxed{A}\bullet\!\!-\!\!\not\!\!\not\!-\!\!\bullet\boxed{B}$ | not both A and B | $\neg\mathbf{F}A \vee \neg\mathbf{F}B$ |

stitch a wound (nor give a tetanus vaccine) before disinfecting the wound. This does not mean that after disinfection we have to stitch a wound (for example, if it is not severe), but only that we may not stitch before disinfection. The response constraint on the other hand means that after executing the source, we have to execute the target. In our example, we must prescribe antibiotics after stitching a wound. It is perfectly acceptable to execute the source multiple times and the target only once, for example stitching a wound two or more times and only prescribe antibiotics once. The non co-existence constraint models that only one of the connected tasks can be executed (but none of them has to be). In our example, we cannot both give a tetanus vaccine and prescribe an antibiotics, maybe because the effect of the vaccine is diminished or voided by the antibiotics.

We notice that we can rely on the informal semantics to understand local aspects of a model. We also see that at no point do we talk about any explicit execution order: *anything not explicitly forbidden is allowed.* A DECLARE model with no constraints allows any execution of any task in the model. In the example in Fig. 4, after performing patient registration and wound disinfection, the process allows executing any task in the model. This allows a lot of freedom, which is often useful in a highly dynamic process, such as a medical process, or in a process which is not very well-known. The dynamic behavior is present in processes where a lot of possibilities are available and there is no obvious best choice. Processes can be partially unknown either because they are so complex nobody fully understands the entire process, or because they are early in the specification phase. Using a declarative approach allows modelers to only specify known constraints, such as incompatibilities between two treatments or a required follow-up to one treatment, and not worry about concrete global execution order.

The formal semantics of DECLARE is given using (finite) linear temporal logic (LTL). This yields a compact syntax, and an abstract and well-understood semantical foundation. More interesting is the fact that finite LTL can be translated into finite automata [24, 47]. These automata can be used to analyze the process and to provide enactment. As the modeler does not explicitly specify an execution order, the system has to figure out what constitutes a legal execution. This is done by instantiating a constraint template for each constraint. Instantiation comprises of taking the formal semantics for each constraint (Table 1) and replacing the parameters with actual tasks, obtaining for example $\bar{\mathbf{X}}$(Receive Patient) for the init constraint. We can get a specification of the full semantics of the entire system by taking the conjunction of all instantiations of constraint templates, and translating it to a finite automaton with task names as transition labels. We can use this automaton to enact the system by following states in the automaton and only allowing tasks that lead to states from which an accepting state is reachable.

The translation from LTL to automata is exponential in the size of the formula given, so it may seem it would be better to consider each constraint in isolation. This is not sufficient, however, because DECLARE models may have implicit choices that are not immediately obvious. In our example in Fig. 4 we have an explicit choice between Prescribe Antibiotics and Give Tetanus Vaccine, but we actually have an implicit choice in that system as well. If we consider the trace Receive Patient;Disinfect Wound;Give Tetanus Vaccine;Stitch Wound, we see that we do not irreparable violate any constraint, but neither is it possible to arrive at a situation where all constraints are satisfied at the same time. After executing the trace, the response from Stitch Wound to Prescribe Antibiotics is not satisfied, because we have stitched the wound without prescribing antibiotics yet. We can satisfy this constraint by prescribing antibiotics, but this would violate the non co-existence between Give Tetanus Vaccine and Prescribe Antibiotics. The DECLARE system does this analysis and will not allow an execution trace with both Stitch Wound and Give Tetanus Vaccine. In [31] we describe how this can be efficiently represented in a single automaton, which keeps track of the status of individual constraints and of the entire process, and in [47] we give an efficient means of constructing this automaton.

A very strong point about declarative workflows is that models are less concerned with the actual execution, and hence more modular. In our previous execution trace, we said that the DECLARE system would prevent executing both Give Tetanus Vaccine and Stitch Wound. This can be overridden by the user, however. For example maybe an initial assessment concludes that stitches are not necessary and instead gives a tetanus shot. This is subsequently readdressed and now stitching is deemed necessary. The system can then say that either the non co-existence constraint or the response will be violated by this action, and authorized personnel can choose to proceed, ignoring a constraint in the process. This effectively is a migration from one model to another (with fewer constraints). As removing constraints never removes allowed behavior, this is guaranteed to be successful. We can also do migration in cases where we add new constraints as long as the added constraint is not violated and in conflict with another constraint (or the conflicting constraints are removed). This means that a declarative model not only allows flexibility by naturally producing permissive models, but also by allowing deviations from the model and even changing the model on-the-fly [35].

The automaton can obviously also be used a-posteriori to do conformance checking of models with respect to historical data, or to check for and remove dead parts of models [29, 33]. DECLARE models can be mined automatically by systematically instantiating all constraint templates and checking them against the historical log [32].

## 2.4 Functional Workflows (iTasks)

Functional workflow specifications arise from functional programming. This means that functional workflows automatically inherit properties of functional languages, importantly explicit flow of data and ability to efficiently execute the resulting specifications in complicated computing environments, including parallel and distributed execution.

The iTask system (iTASKS) augments a standard functional programming language, CLEAN [36], with connectives useful for workflow specification [37, 39].[10] The basic unit in an iTASKS process is a *task*, which is a basic type describing a process to be executed. iTASKS builds on the functional idea of inductively defining values in terms of basic values and composite values. A basic task value would be inputting an integer into a field and a composite task could be sequencing tasks or making a choice. The sequence is already well-known to procedural programmers but used less in functional programming. The choice is similar to a conditional in traditional programming, but iTASKS has a general choice operator, which can be specialized (and is by default) to provide semantics of XOR, AND, OR, and parallel splits as well as more specialized splits, for example a 2-out-of-3 split.

An iTASKS process consists of specification of the data types used and a process specification. In Listing 1, we see a simple patient registration process for a hospital. The example illustrates both the major weakness and major strength of the paradigm: the code is very verbose for a simple descriptive model, but compact for a full-features implementation of a system for executing the process. The process comprises data definitions (ll. 4-15), helper functions (ll. 17-23), and a process description (ll. 25-53). The data definitions describe a patient record (ll. 4-8) and an insurance record (ll. 10-13). In addition, we have an instruction to the system to generate a full implementation of these from the data specification (l. 15). The returnV function (ll. 17-18) is used for technical reasons, and the hasName and isInsured functions (ll. 20-23) are simple predicates on patients. Lines 25-28 set up the main process, which consists of starting the

[10] iTASKS and CLEAN can be downloaded from `itasks.cs.ru.nl`.

**Listing 1.** iTasks model of patient registration.

```
implementation module PatientRegistration
import iTasks

:: Patient =
  { name        :: String
  , dateOfBirth :: Maybe Date
  , insurance   :: Bool
  }

:: InsuranceInfo =
  { company       :: String
  , insuranceNumber :: Int
  }

derive class iTask Patient, InsuranceInfo

returnV :: (TaskValue a) -> Task a | iTask a
returnV (Value v _) = return v

hasName (Value { name, dateOfBirth, insurance } _) = name <> ""
hasName _ = False

isInsured { name, dateOfBirth, insurance } = insurance

Start :: *World -> *World
Start world = startEngine (manageWorklist [mainProcess]) world

mainProcess = workflow "New␣Patient" "Handle␣a␣patient" handlePatient

enterPatient :: Task Patient
enterPatient = enterInformation "Enter␣patient␣information" []

enterInsurance :: Task InsuranceInfo
enterInsurance = enterInformation "Enter␣insurance␣information" []

treat :: Patient -> (Task Patient)
treat patient =
    viewInformation ("Treat␣Patient", "Treating␣Patient") [] patient

showInsurance :: InsuranceInfo -> (Task InsuranceInfo)
showInsurance insurance =
    viewInformation ("Insurance", "Insurance␣Details") [] insurance

handlePatient :: Task (Patient, Maybe InsuranceInfo)
handlePatient =
    enterPatient
    >>* [OnAction (Action "Continue") hasName returnV]
    >>= \patient ->
            if (isInsured patient)
            ((treat patient ~~&&~~
              (enterInsurance >>= showInsurance))
             >>= \(p, i) -> return (p, Just i))
            (treat patient >>= \p -> return (p, Nothing))
```

task handlePatient (ll. 44-53) which is a composite task using the primitive tasks in lines 30-42. The primitive tasks comprises two tasks for inputting information for each of the two defined data types (ll. 30-34) and two tasks for displaying information (ll. 36-42). The treat task is a simplified placeholder version of a full treatment. The handlePatient task is by far the more complicated one and shows some of the task combinators supported by iTasks. First we enter patient information (l. 46). The >>* combinator allows us to add ways to proceed the workflow; line 47 produces a Continue button that is enabled when the hasName predicate holds for the patient record entered. In that case, the patient record is passed on. The >>= combinator allows us to pass a result from one task to the next in a sequence. The combinator expects a function taking the result of the previous task as the first parameter, so we make an anonymous function (l. 48). We then use a common if statement (ll. 49-53) to branch according to whether the patient has an insurance or not. If the patient does (ll. 50-52) we start treating them (l. 50) in parallel (-&&-) with inputting and subsequently showing insurance information (l. 51). Finally, we return the treated patient and their insurance information (l. 52). If a patient does not have insurance, they are treated and returned without insurance information (l. 53).

We notice that iTasks is very explicit about data flow and types. We explicitly type all tasks. This is used by the system to automatically generate a (web-based) user interface. We need to explicitly pass and compute data, and a strong type system prevents errors. The explicit passing of information is more verbose than the previously discussed paradigms, but the process oriented combinators makes it very simple to do things that are normally very complicated in programming, such as the parallel split in ll. 50-51 of Listing 1. We can easily change that to an XOR split, OR split, or sequence as long as we preserve the types. Similarly, the treat task (ll. 36-38) is very abstract and would be detailed further in a more elaborate implementation. As long as types are preserved, we can do that without changing the rest of the model. Finally, it is possible to change the type of, e.g., the patient record (ll. 4-8) and add new fields if necessary without changes to most of the tasks; for example, the enterPatient task (ll. 30-31) explicitly state the type but polymorphism and introspection makes the user-interface automatically adapt to the changed type. In our example we need to change the signature of the hasName and isInsured functions (ll. 20-23), but this can be avoided by using a slightly more verbose syntax we have avoided here for simplicity.

*The fact that* iTasks *is realized as a combinator library in a real programming language, makes it possible to easily extend with features not normally available in workflow languages.* For example, modeled processes can natively communicate over the network, making it possible to easily invoke remote services. Another advantage is that there is no gap between the model and the implementation; they are really one and the same in iTasks. Using a general-purpose functional language also makes it possible to use traditional functional techniques, such as making higher-order tasks. In fact, iTasks combinators are just higher order tasks and can be used to create new composite tasks. This makes it easy to add new combinators as needed. An elegant consequence of this is that the process of choosing a task to work on and maintaining a worklist can be considered tasks as well, and in iTasks they are realized as such. The manageWorklist used in line 26 is actually just a task taking a list of (wrapped) tasks and allowing a user to pick which to execute. This makes it possible to customize how tasks are presented and chosen.

## 3 Comparison and Market Analysis

After presenting the distinguishing features of YAWL, BPM|one, Declare, iTasks, and the corresponding paradigms, we aim to identify differences and commonalities. Table 2 shows our main findings. The characteristics used in Table 2 are based on topics frequently discussed in BPM literature. See for example the survey paper [3] which identifies twenty main BPM use cases based on an analysis of all 289 papers published at the main BPM conference series in a ten year period [7], [11], [19], [5], [22], [17], [20], [18], [27], and [42]. The list of characteristics used in Table 2 is far from complete. Nevertheless, we feel that the list is representative for our high-level comparison and discussion of the four different paradigms. In the remainder we discuss these findings and also provide an analysis of the BPM/WFM market.

### 3.1 Basic Characteristics of the Different Paradigms

Table 2 characterizes the four workflow paradigms and tools using three characteristics: *focus*, degree of *coupling*, and *extendibility*. Procedural languages like YAWL are driven by control-flow [9]. Case handling systems like BPM|one are data driven, i.e., the moment a data element gets a value or changes value, the state of the process instance is recomputed [15]. Declare allows for any behavior unless forbidden through constraints [12]. iTasks specifies the desired behavior in terms of a functional program extended with special workflow operators [39]. YAWL and Declare use service-orientation and a clear separation between data and control-flow to decouple different perspectives. In iTasks and BPM|one these are deliberately coupled, e.g., control-flow and data are intertwined to provide additional support and expressiveness. Declare and iTasks are extendible, i.e., the language can be extended by adding new constraint templates [12] or combinators [39]. This is not possible in contemporary procedural workflow and case handling systems.

### 3.2 Flexibility Support

Table 2 list four types of flexibility. These originate from the classification in [44].

*Flexibility by definition* is the ability to incorporate alternative execution paths within a process definition at design time such that selection of the most appropriate execution path can be made at runtime for each process instance.

**Table 2.** Comparison of paradigms.

| | Procedural (YAWL) | Case Handling (BPM\|ONE) | Declarative (DECLARE) | Functional (ITASKS) |
|---|---|---|---|---|
| **Characteristics** | | | | |
| Focus | control-flow[a] | data dependencies[b] | tasks and constraints[c] | functional program[d] |
| Coupling of perspectives | decoupled | tight | decoupled | tight |
| Extendible | | | √[e] [12] | √ [37, 39] |
| **Flexibility** | | | | |
| Definition | √[f] [9, 26] | √[g] [15] | √[h] [12] | √[i] [37, 39] |
| Deviation | | √[j] [15] | √[k] [12, 35] | |
| Underspecification | (√)[l] [4, 16, 26] | | (√)[m] [4, 12] | √[n] [37, 39] |
| Change | (√)[o] | | √[p] [35] | √[q] [38] |
| **Analysis** | | | | |
| Verification | √ [13, 26] | | √ [12, 47] | (√)[r] |
| Performance analysis | √ [13, 14, 26] | | √[s] | |
| Process discovery | √ [2, 26] | | √ [32] | |
| Conformance checking | √ [2] | | √ [29, 31] | |

[a] Token-based semantics (like playing the token game on a Petri net).
[b] Available data objects and their values determine the state.
[c] Anything that is not explicitly forbidden through some combination of constraints is allowed.
[d] Extendible set of combinator functions are used to specify the flow of work.
[e] Can be achieved by adding a template and LTL semantics.
[f] Supports XOR/AND/OR-splits and joins, cancelation regions, deferred choice, multiple instance tasks, etc.
[g] Supports XOR/AND/OR-splits and joins next to data-driven behavior.
[h] Anything is allowed unless explicitly forbidden.
[i] Core combinators can be used to express basic workflow patterns.
[j] Authorized users can deviate by skipping and redoing tasks.
[k] Users can choose to ignore non-mandatory constraints.
[l] Supported using worklets in YAWL but not a common feature for procedural languages.
[m] Can defer the task execution to other tools.
[n] Selecting a task is itself a task.
[o] Not supported by YAWL, but there are systems like ADEPT that support such changes [40, 41].
[p] Possible to migrate cases that do not violate constraints in the new model. Migration is postponed if needed.
[q] Change is viewed as a type-safe replacement of one task function by another one.
[r] Static analysis and common program analysis.
[s] Under development (combining DECLARE and CPN TOOLS).

For example, an XOR-split defined at design time adds the ability to select one or more activities for subsequent execution from a set of available activities. Parallelism defined at design time leaves the actual ordering of activities open and thus provides more flexibility than sequential routing. All WFM/BPM systems support this type of flexibility. However, declarative languages make it easier to defer choices to runtime.

The classical workflow patterns mentioned earlier [10] can be viewed as a classification of "flexibility by definition" mechanisms for procedural languages. For example, the "deferred choice" pattern [10] leaves the resolution of a choice to the environment at runtime. Note that a so-called "flower place" in a Petri net, i.e., a place with many transitions that have this place as only input and output place, provides a lot of flexibility.

*Flexibility by deviation* is the ability for a process instance to deviate at runtime from the execution path prescribed by the original process without altering the process definition itself. The deviation can only encompass changes to the execution sequence for a specific process instance, and does not require modifications of the process definition. Typical deviations are *undo*, *redo*, and *skip*.

The BPM|ONE system of Perceptive Software is a system that provides various mechanisms for deviations at runtime. The case handling paradigm [15] supported by BPM|ONE allows the user to skip or redo activities (if not explicitly forbidden and assuming the user is authorized to do so). Moreover, data can be entered earlier or later because the state is continuously recomputed based on the available data. DECLARE supports flexibility by deviation through *optional* constraints.

*Flexibility by underspecification* is the ability to execute an incomplete process specification, i.e., a model that does not contain sufficient information to allow it to be executed to completion. An incomplete process specification contains one or more so-called *placeholders*. These placeholders are nodes which are marked as underspecified (i.e., "holes" in the specification) and whose content is specified during the execution of the process. The manner in which these placeholders are ultimately enacted is determined by applying one of the following approaches: *late binding* (the implementation of a placeholder is selected from a set of available process fragments) or *late modeling* (a new process fragment is constructed in order to complete a given placeholder). For late binding, a process fragment has to be selected from an existing set of fully predefined process fragments. This approach is limited to selection, and does not allow a new process fragment to be constructed. For late modeling, a new process fragment can be developed from scratch or composed from existing process fragments.

In the context of YAWL [9], the so-called *worklets* approach [16] has been developed which allows for late binding and late modeling. Late binding is supported through so-called "ripple-down rules", i.e., based on context information the user can be guided to selecting a suitable fragment. In [43] the term "pockets of flexibility" was introduced to refer to the placeholder for change. In [25] an explicit notion of "vagueness" is introduced in the context of process modeling.

The authors propose model elements such as arc conditions and task ordering to be deliberately omitted from models in the early stages of modeling. Moreover, parts of the process model can be tagged as "incomplete" or "unspecified". In iTasks selecting a task is itself a task [37, 39]. This can be used to support late binding.

*Flexibility by change* is the ability to modify a process definition at run-time such that one or all of the currently executing process instances are migrated to a new process definition. Changes may be introduced both at the process instance and the process type levels. A *momentary change* (also known as change at the instance level) is a change affecting the execution of one or more selected process instances. An example of a momentary change is the postponement of registering a patient that has arrived to the hospital emergency center: treatment is started immediately rather than spending time on formalities first. Such a momentary change performed on a given process instance does not affect any future instances. An *evolutionary change* (also known as change at the type level) is a change caused by modification of the process definition, potentially affecting all new process instances. A typical example of the evolutionary change is the redesign of a business process to improve the overall performance characteristics by allowing for more concurrency. Running process instances that are impacted by an evolutionary or a momentary change need to be handled properly. If a running process instance is transferred to the new process, then there may not be a corresponding state (called the "dynamic change bug" in [23]).

Flexibility by change is challenging and has been investigated by many researchers. In the context of the ADEPT system, flexibility by change has been examined in detail [40, 41]. This work shows that changes can introduce all kinds of anomalies (missing data, deadlocks, double work, etc.). For example, it is difficult to handle both momentary changes and evolutionary changes at the same time, e.g., an ad-hoc change made for a specific instance may be affected by a later change at the type level. The declarative workflow system Declare has been extended to support both evolutionary and momentary changes [35] thus illustrating that a declarative style of modeling indeed simplifies the realization of all kinds of flexibility support. In [38] it is shown that replacing a task can be seen as a type-safe replacement of one pure function by another one. The iTasks type system ensures that the values passed between task have the correct type in the initial workflow as well as after any number of changes in this workflow. Note that such changes are more restrictive than in some of procedural and declarative approaches, e.g., the degree of concurrency can not be changed other than by replacing the whole subprocess. Moreover, there should be a dedicated user interface to support such changes. Otherwise, it is unrealistic to assume that end-users can define new functions on-the-fly. However, this holds also for most other approaches.

## 3.3 Analysis Support

Table 2 illustrates that procedural and declarative approaches are supported by a range of analysis techniques. For example, procedural workflow languages benefit from the verification [1, 13, 26], simulation [13, 14, 26], and process mining techniques [2, 26] developed for Petri nets. To apply these results to industry-driven languages such as BPMN, UML activity diagrams, EPCs, and BPEL conversions are needed. Sometimes there conversions need to make abstractions (e.g., ignoring data or time). Nevertheless, it is fair to say that most analysis techniques are tailored towards these procedural workflow languages.

In recent years, various analysis techniques have been developed for declarative languages like Declare [12, 24, 29, 31–33, 35, 47]. These techniques heavily rely on the fact that the semantics of Declare are defined in terms of LTL and that there are various ways to translate Declare constraints into automata.

BPM|one and iTasks are more implementation-oriented providing hardly any dedicated analysis support.[11]

## 3.4 Overview of BPM/WFM Market

Given the very different styles of process automation that we distinguished and explained in the previous sections, it seems worthwhile to reflect on their use in practice. One way of doing so is to consider what the dominating paradigms are that commercial vendors of BPM systems have adopted in their products. There are more than 100 BPM vendors active at this point of writing, which makes a full consideration of all existing products infeasible. Instead, we will rely on a subset of these as they are listed in the so-called Magic Quadrant on BPM systems provided by Gartner [45]. Gartner is a market analyst that has been following the BPM market space for a number of years and its reports are highly influential in how companies decide on their selection of products in the BPM domain. Specifically, we will rely on their 2010 version of this quadrant; it is provided in Figure 5.

In the diagram, 27 BPM suites of 25 different vendors are displayed. While a BPM suite arguably encompasses more functionality than a BPM system does, it is safe to say that process automation is at its core.[1]

Two dimensions are used to differentiate the various offerings in the Magic Quadrant. The 'ability to execute' refers to the presence of a particular product in the market place, while the 'completeness of vision' reflects the analyst's view on the innovation and breadth of the offering. Gartner evaluated over 60 BPM vendors to select the top performers with respect to these criteria [45]. For these reasons, we consider this overview as useful for profiling both popular and best-in-class BPM systems.

We conducted a light-weight evaluation of all the products that are described in the Magic Quadrant, which comprised of the following steps. First, we checked whether the products were still available in the market place and in what form. By doing so, we established that various offerings changed hands because of

[11] Note that simulation and process discovery are supported for the procedural parts of BPM|one, but not for the parts specific for case handling.

[1] Note that for this reason we have not used Gartner's 2012 Magic Quadrant on intelligent BPM suites in which the analytical capabilities play a much bigger role.

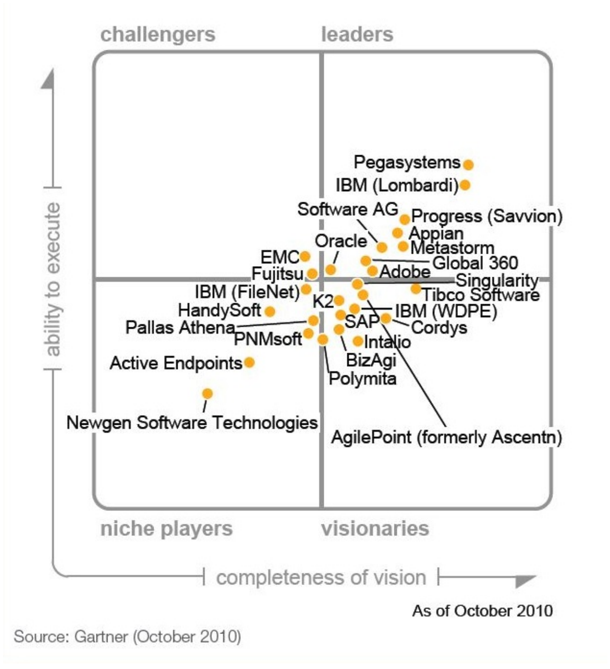

**Fig. 5.** Gartner's Magic Quadrant on the market of BPM Suites.

acquisitions. Also, vendors decided to integrate multiple solutions they offered. Specifically, OpenText acquired both Metastorm and Global 360, Perceptive Software acquired Pallas Athena, Singularity is now Kofax, and Polymita is acquired by Red Hat. OpenText integrated its two BPM products into one offering; IBM integrated its Lombardi and WDPE offerings (its Filenet product is still a separate product). As a result, of the original 27 products in the Magic Quadrant, 25 can still be considered to be active.

The second step consisted of the actual evaluation. We consulted the online document information of each of the vendors on their products, specifically related to their modeling approach and tool set. We also studied the samples that vendors provided of actual models of business processes as specified with their tools. This approach is light-weight in the sense that there was no interaction with experts of the vendors and that no hands-on use of these tools took place.

The evaluation led to the insight that *of all 25 products, 23 of these dominantly follow a procedural approach for specifying and enacting workflows.* Two of the 25 products can be said to adhere to the case management paradigm, i.e. Perceptive Software's BPM|ONE and OpenText's BPM solution. No products can be clearly said to support declarative or functional workflows.

There are two interesting side-notes to make. First of all, the adherence to the procedural approach to workflow modeling seems closely aligned with the widely prevalent support among vendors for BPMN [34]. As discussed in

Section 2.1, the BPMN notation essentially supports a procedural approach, in which explicit paths are modeled in a sequential manner. The BPMN standard does cover a so-called *Ad Hoc Process*, in which the relations between activities is less prescribed. The current development efforts by OMG to specify the Case Management and Modeling Notation (CMNN) seems to head towards the use declarative rules and constraints to pinpoint the semantics of this element. As such, the popularity of BPMN among BPM vendors may pave the way to the uptake of more declarative aspects in workflow specifications, although this is a tentative development.

Secondly, case management is a paradigm that is claimed by many vendors as a feature of their offerings, despite their dominant adherence to the procedural paradigm. Such claims mostly build on the notion that workflows provide a lifecycle of a particular type of case and that a *document-centric view* on such cases is provided with their products. Despite the value of this idea, it is a far cry from the paradigm that we described in Section 2.2. Closely related to the previous observation is that both the products that adhered to case management as the dominant paradigm also fully support a procedural approach.

In conclusion, the procedural paradigm can be said to be the overly dominant one in the marketplace. In second place, but a long way behind, is the case management paradigm. Interestingly, case management is perceived as an attractive idea by many vendors although it is never offered as a fully stand-alone approach (i.e. totally ruling out the use of procedural workflows). Declarative and functional approaches are not spotted in the marketplace. Since the different approaches are complementary, we hope to see combined approaches in commercial systems in the future.

## 4 A Matter of Taste?

In this paper we compared different workflow paradigms that all played a role in the STW project "Controlling Dynamic Real Life Workflow Situations with Demand Driven Workflow Systems" where we collaborated with Rinus Plasmeijer and his team. In our view none of the four paradigms (procedural, case handling, declarative, and functional) is superior (or inferior). All emphasize different aspects. For example, analysis techniques ranging from verification and performance analysis to process discovery and conformance checking are best developed for procedural languages (e.g., Petri nets). However, recently, also many analysis techniques have been developed for DECLARE. Case handling systems like BPM|ONE and functional language extensions like iTASKS provide little support for analysis. Instead, these approaches concentrate on the development of data- and process-centric information systems. Both BPM|ONE and DECLARE focus on offering flexibility to end-users. iTASKS provides a different kind of flexibility: workflows can be reused and changed easily. In fact, new combinators can be added making the language extensible (like DECLARE). The iTASKS language would benefit from a graphical front-end to make it more understandable. However, such a graphical front-end could reduce expressiveness and limit flexibility at design time. iTasks is very fast compared to existing systems; it has the pro's and con's of a programming language.

Each of the four approaches is beautiful in its own way and has a different appeal to it (analysis, reuse, flexibility, maintainability, etc.). For example, different forms of flexibility are possible and it is not realistic to assume a single language that suits all purposes. Partly, the choice of language is also a matter of taste. Fortunately, one can combine different approaches as discussed in [4]. A task in one language may correspond to a subprocess in another language. This way different styles of modeling and enactment may be mixed and nested in any way appropriate.

**Acknowledgements**

This work was supported by the Basic Research Program of the National Research University Higher School of Economics (HSE).

## References

1. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.
2. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer-Verlag, Berlin, 2011.
3. W.M.P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, Article ID 507984, doi:10.1155/2013/507984:1–37, 2013.
4. W.M.P. van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg. Flexibility as a Service. In L. Chen, editor, *Database Systems for Advanced Applications (DASFAA 2009)*, volume 5667 of *Lecture Notes in Computer Science*, pages 319–333. Springer-Verlag, Berlin, 2009.
5. W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors. *International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2005.
6. W.M.P. van der Aalst, P. de Crom, R. Goverde, K.M. van Hee, W. Hofman, H. Reijers, and R.A. van der Toorn. ExSpect 6.4: An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, Berlin, 2000.
7. 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.
8. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2004.
9. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
10. 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.

11. 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*. Springer-Verlag, Berlin, 2003.
12. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 23(2):99–113, 2009.
13. W.M.P. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net Oriented Approach*. MIT press, Cambridge, MA, 2011.
14. W.M.P. van der Aalst, C. Stahl, and W. Westergaard. Strategies for Modeling Complex Processes using Colored Petri Nets. In K. Jensen, K. Wolf, W.M.P. van der Aalst, G. Balbo, and M. Koutny, editors, *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2013.
15. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
16. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In R. Meersman and Z. Tari et al., editors, *On the Move to Meaningful Internet Systems 2006, OTM Confederated International Conferences, 14th International Conference on Cooperative Information Systems (CoopIS 2006)*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer-Verlag, Berlin, 2006.
17. G. Alonso, P. Dadam, and M. Rosemann, editors. *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2007.
18. U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors. *International Conference on Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2009.
19. J. Desel, B. Pernici, and M. Weske, editors. *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.
20. M. Dumas, M. Reichert, and M.C. Shan, editors. *International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2008.
21. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer-Verlag, Berlin, 2013.
22. S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors. *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2006.
23. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic Change within Workflow Systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
24. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Proceedings of the 16th IEEE international conference on Automated software engineering (ASE'01)*, pages 412–416. IEEE Computer Society, 2001.
25. T. Herrmann, M. Hoffmann, K.U. Loser, and K. Moysich. Semistructured Models are Surprisingly Useful for User-Centered Design. In G. De Michelis, A. Giboin, L. Karsenty, and R. Dieng, editors, *Designing Cooperative Systems (Coop 2000)*, pages 159–174. IOS Press, Amsterdam, 2000.

26. A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer-Verlag, Berlin, 2010.
27. R. Hull, J. Mendling, and S. Tai, editors. *International Conference on Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2010.
28. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
29. M. de Leoni, F.M. Maggi, and W.M.P. van der Aalst. Aligning Event Logs and Declarative Process Models for Conformance Checking. In A. Barros, A. Gal, and E. Kindler, editors, *International Conference on Business Process Management (BPM 2012)*, volume 7481 of *Lecture Notes in Computer Science*, pages 82–97. Springer-Verlag, Berlin, 2012.
30. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
31. F. Maggi, M. Montali, M. Westergaard, and W. van der Aalst. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer-Verlag, Berlin, 2011.
32. F.M. Maggi, R.P. Jagadeesh Chandra Bose, and W.M.P. van der Aalst. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In J. Ralyte, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *International Conference on Advanced Information Systems Engineering (Caise 2012)*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer-Verlag, Berlin, 2012.
33. M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1):1–62, 2010.
34. OMG. Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03, 2011.
35. M. Pesic, M. H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer-Verlag, Berlin, 2007.
36. R. Plasmeijer. CLEAN: A Programming Environment Based on Term Graph Rewriting. *Electronic Notes in Theoretical Computer Science*, 2:215–221, 1995.
37. R. Plasmeijer, P. Achten, and P. Koopman. iTasks: Executable Specifications of Interactive Workflow Systems for the Web. *SIGPLAN Notices*, 42(9):141–152, 2007.
38. R. Plasmeijer, P. Achten, P. Koopman, B. Lijnse, T. van Noort, and J. van Groningen. iTasks for a change: Type-safe run-time change in dynamically evolving workflows. In *Proceedings of the 20th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pages 151–160, New York, NY, USA, 2011. ACM.
39. R. Plasmeijer, B. Lijnse, S. Michels, P. Achten, and P. Koopman. Task-Oriented Programming in a Pure Functional Language. In *Proceedings of the 14th symposium on Principles and Practice of Declarative Programming*, pages 195–206, New York, NY, USA, 2012. ACM.

40. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
41. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
42. S. Rinderle, F. Toumani, and K. Wolf, editors. *International Conference on Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2011.
43. S. Sadiq, W. Sadiq, and M. Orlowska. Pockets of Flexibility in Workflow Specification. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001)*, volume 2224 of *Lecture Notes in Computer Science*, pages 513–526. Springer-Verlag, Berlin, 2001.
44. H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst. Process Flexibility: A Survey of Contemporary Approaches. In J. Dietz, A. Albani, and J. Barjis, editors, *Advances in Enterprise Engineering I*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer-Verlag, Berlin, 2008.
45. J. Sinur and J. Hill. Magic Quadrant for Business Process Management Suites, Gartner RAS Core Research Note G00205212. `www.gartner.com`, 2010.
46. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.
47. M. Westergaard. Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 83–98. Springer-Verlag, Berlin, 2011.