Chapter 1
Introduction

Wil van der Aalst, Michael Adams, Arthur ter Hofstede, and Nick Russell

1.1 Overview

The area of Business Process Management (BPM) has received considerable attention in recent years due to its potential for significantly increasing productivity and saving cost. In BPM, the concept of a process is fundamental and serves as a starting point for understanding how a business operates and what opportunities exist for streamlining its constituent activities. It is therefore not surprising that the potential impact of BPM is wide-ranging and that its introduction has both managerial as well as technical ramifications.

While benefits can be derived from BPM even when its application is restricted to what can be described as “pen-and-paper” exercises, such as the visualization of business process models in order to discuss opportunities for change and improvement, there is potentially much more to be gained if such an analysis serves as the blueprint for subsequent automation of key business processes. In the area of Business Process Automation (BPA), sometimes referred to as workflow management, precise business process descriptions are used to guide the performance of business activities. Work is delivered to selected resources, which can be either humans or software applications, when it needs to be executed. Progress can be monitored and may give rise to the escalation of certain tasks where their deadline has passed or is not likely to be met. Events, such as the completion of a certain task by a certain resource, are logged and the resulting log files can be exploited for analysis purposes, an area of interest in its own right typically referred to as process mining.

Substantial cost and time savings can be achieved through the use of workflow technology. When describing a workflow, which is an executable process, one has to capture all aspects relevant to automation, such as the individual activities (or tasks) and their execution order, data that is to be entered and passed on, and the way resources are involved. By taking this holistic view of a business process and capturing both the tasks and the data involved, it is less likely that inconsistencies arise from data not being entered or updated during the execution of a certain
process. Consider, for example, the case of an employee being granted permission for leave; this should not only result in an email notification to the applicant, but also in an update to the Human Resources records. By capturing both tasks and resources, it is possible to expedite processes using information regarding the current availability and workload of resources. Or, consider the case of a travel application delayed for 5 days, because it ended up in the in-tray of the absent director, while it should have been automatically rerouted to the acting director. Or the case of the sales inquiry that was left unanswered as it was directly addressed to a sales representative who had since left the company. A task allocation on the basis of roles rather than individuals would have avoided this problem. As a consequence of the explicit representation of tasks and their chronological dependencies, as well as the involvement of resources in the execution of these tasks, it is easier to adapt business processes in order to react in a timely manner to environmental changes, for example, market fluctuations or legislative adaptations. Instead of having to make changes somewhere deep in application code, these changes can be made at the specification level. Analysis and simulation support may help decide whether these changes satisfy certain correctness criteria or are likely to have their intended effect before they are actually deployed. Monitoring capabilities provide scope for rapid detection of problems and subsequent escalation, while post-execution log analysis (i.e., process mining) can provide a solid basis for process improvement.

In the field of BPM, it is recognized that business processes go through various stages of the so-called BPM life-cycle, cf. Fig. 1.1. Business processes start this life-cycle when they are created, either from scratch or through configuration of an existing model. This corresponds to the process (re)design phase in Fig. 1.1. The business process is subsequently implemented by configuring the corresponding information system. This system configuration phase may require substantial implementation efforts or may be a simple selection step. This all depends on the underlying technology and on how much customization is needed. Then the process can be executed and monitored in the process enactment and monitoring phase. Finally, in the diagnosis phase one can learn from the running process and use this as input for business process improvement. Diagnosis of the actual process

![Fig. 1.1 The BPM life-cycle](image-url)
execution may result in its adaptation. Adaptation may involve some modeling, which then may lead to a new or revised version of the business process for which the life-cycle starts again. Because of the automated support for managing the business process life-cycle, businesses can rapidly adapt to their ever-changing environment.

Much has been written about business processes; however, there is a lack of consensus about how they are best described for the purposes of analysis and subsequent automation. This has resulted in a plethora of approaches for capturing business processes, though not all are intended to support direct automation. There are two main reasons to which this situation can be attributed:

- Business processes can be complex. Their specification may involve capturing complex ordering dependencies between tasks and complex resourcing strategies. Process modeling languages tend to lack the concepts to be able to deal with the broad range of requirements one may encounter when trying to precisely capture business scenarios.
- Standardization efforts in the field have essentially failed. One may argue that this is the result of the standardization processes being partly driven by vested business interests. Whatever the reason, it is clear that today’s standards lack widespread adoption and suffer from all kinds of technical problems.

The inherent complexity of business processes and the question of what fundamental concepts are necessary for business process modeling gave rise to the development of a collection of workflow patterns. These patterns describe process modeling requirements in an implementation independent manner.

In this chapter, it is shown how the workflow language YAWL and its corresponding system emerged from the Workflow Patterns Initiative. However, before doing so, the role of models in BPM is discussed and some of the standard approaches are reviewed.

### 1.2 On the Role of Models in BPM

Models can serve different purposes in BPM.

First of all, models may aim at providing insight. When developing or improving an information system, it is important that the different stakeholders get insight into the processes at hand and the way that these processes can or should be supported. Models can be used to discuss requirements, to support design decisions, and to validate assumptions. Moreover, the modeling process itself typically provides new and valuable insights, because the modeler is triggered to make things explicit.

Second, models may be used to analyze the system and/or its processes. Depending on the type of model, particular types of analysis are possible or not. In the context of BPM, analysis may focus on the business processes or on the information system itself. For example, the performance of a system (e.g., response times) is not the same as the performance of the processes it supports. Traditionally, most
techniques used for the analysis of business processes originate from operations research. Students taking courses in operations management will learn to apply techniques such as simulation, queueing theory, and Markovian analysis. The focus mainly is on performance analysis and less attention is paid to the correctness of models. However, verification is needed to check whether the resulting system is free of logical errors. Many process designs suffer from deadlocks and livelocks that could have been detected using verification techniques. Notions such as soundness can be used to verify the correctness of the systems.

Finally, models can be used to enact processes. In the context of a BPM system, models are often used for enactment, that is, based on a model of the process, the corresponding runtime support is generated. In a workflow management system, a model of a process suffices to generate the corresponding system support. In other environments, the set of processes is often hard-coded. For example, although ERP systems like SAP have a workflow engine, most processes are hard-coded into the system and can only be changed by programming or changing configuration parameters. As a result, modifications are either time-consuming (because of substantial programming efforts) or restricted by the set of predefined configuration parameters.

Figure 1.2 shows another view on the role of (process) models. Models can be used to model and analyze operational processes without explicitly considering the information system. Consider, for example, the use of business process simulation in environments where Information Technology (IT) plays only a minor role. However, models can also be used to configure and/or implement systems. The basic idea of workflow technology is to generate systems on the basis of models. Figure 1.2 emphasizes the role of event logs. Information systems record events and this information can be used when designing or analyzing processes. In fact, workflow management systems provide excellent facilities for logging events at

Fig. 1.2 On the role of (process) models in a BPM setting
the business process level. This information can be used to automatically discover models based on frequent patterns. Moreover, if a model is given, then the conformance of the process with respect to this model can be measured. The latter is interesting when analyzing compliance and process flexibility. Both conformance checking and process discovery are part of the process mining domain.

1.3 BPM Standard Approaches

Over the years, there have been many approaches toward the specification of business processes. Many BPM tools supported their own languages and it was often unclear how these languages compared. Over time a number of standards and/or widely used approaches emerged, and we will briefly look at some of the more important ones in this section.

One of the first attempts to define a standard approach to the specification of executable business processes was the XML Process Definition Language (XPDL) 1.0, defined in the nineties by the Workflow Management Coalition (WFMC), an industry body promoting the spread and further development of workflow technology. XPDL was intended to facilitate interoperability between workflow environments. The language offered a minimal set of generally occurring routing constructs such as various splits and joins, and these were defined in natural language. Due to this minimalist approach and to the fact that various interpretations of even these basic constructs was possible, the goal of interoperability was not achieved and gradually XPDL 1.0 became irrelevant.

In 2003, the Business Process Execution Language (BPEL) was proposed. This language combined Microsoft’s XLANG and IBM’s Web Services Flow Language (WSFL) and is therefore a language that marries two fundamentally different approaches to the specification of executable business processes. Generally speaking, BPEL is a block-structured language where business processes are specified in terms of self-contained blocks that are composed to form larger, more complex, blocks. However, BPEL is not fully block-structured as it supports the specification of dependencies that cross block boundaries through the use of so-called control links. While BPEL was a clear step forward in terms of its support for the specification of control-flow dependencies, the language provided no support for the involvement of human participants in the execution of business activities. In addition, the language has no graphical representation; specifications have an XML-based depiction.

The Business Process Modeling Notation (BPMN) was introduced to provide an easily understood graphical notation that could serve as a front end to various approaches for the execution of business processes. The language itself is not intended to be directly executable, rather specifications are expected to be transformed to an executable language to achieve their enactment. BPMN provides fairly strong support for the specification of control-flow dependencies and is graph-structured rather than block-structured. Contrary to BPEL, BPMN imposes
no restrictions on the specification of loops, and loops are allowed to have multiple entry and/or exit points. A consequence of this is that mapping a BPMN specification to a corresponding BPEL specification can be a less than trivial matter, and contemporary support tools typically impose restrictions on BPMN diagrams for the purpose of subsequent transformation to BPEL. Similar to BPEL, though slightly better, BPMN does not make much provision for the various ways in which human participants can be involved in the execution of a business process, and given that BPMN does not have a formalization accepted by a standards organization, the interpretation of some of its concepts may vary. Nonetheless, BPMN can be seen as a move in the direction of more expressive languages, and its continued evolution and increased adoption makes it likely to have some longevity. In recognition of this, XPDL has been reinvented and its 2.0 incarnation is an XML serialization of BPMN.

Although not a formal standard, Event-driven Process Chains (EPCs) are a well-known approach to process specification and the notation has been around for over 15 years. EPCs are supported by the ARIS environment, where they are used for business process modeling and simulation. EPCs are not directly executable and they provide a fairly minimal set of control-flow constructs. Extended EPCs augment EPCs with notations for the involvement of participants and the use of data elements. EPCs do not have a formal foundation, though one has been defined by Wil van der Aalst in the late nineties. As he argued that the semantics of the OR join connector are “not clear” and “subject to multiple interpretations,” this formalization does not incorporate this particular connector. It has since been argued that there are inherent semantical problems with this concept in the presence of so-called “vicious circles.”

Another well-known approach to process specification are the Activity Diagrams of the Unified Modeling Language (UML). In their 1.4 incarnation, these were based on statecharts, while in their 2.0 incarnation, their semantics was more inspired by Petri nets. Because of the fact that UML 2.0 activity diagrams do not have a notion that corresponds to the concept of a place in Petri nets, the link between UML 2.0 activity diagrams and Petri nets is rather complicated. This is relevant because of two reasons. First of all, certain business scenarios mixing concurrency and choice cannot be expressed easily. Second, there is no simple and clear semantics. UML activity diagrams are not intended for direct execution and, although a formal semantics for them has been defined, no formalization has been officially endorsed by the Object Management Group (OMG), which is the standardization body behind UML. Furthermore, it seems that in recent years UML Activity Diagrams have been eclipsed by BPMN in the context of the specification of business processes.

1.4 The Workflow Patterns Initiative

The concept of workflow has been around for decades and can be traced back to early work on office automation. Despite its early origins, widespread uptake of workflow management systems in practice and the integration of this technology
Introduction

with other types of systems, generally described as process-aware information systems, did not occur until the mid to late nineties. Although there are technological considerations involved, the lack of a commonly accepted foundation for the specification of executable business processes played a major role in the slow progression of workflow to broad adoption. This resulted in a plethora of approaches, where concepts with similar names could have significant semantic differences. The Workflow Management Coalition (WFMC) failed to provide a standard that was (1) sufficiently precise, and (2) sufficiently expressive. As part of its definition of “interface 1,” it provided natural language definitions of a number of commonly occurring workflow concepts. These definitions led to a situation where vendors could legitimately claim to abide by the WFMC standard, even though their interpretation of these concepts was fundamentally different and could lead to the same workflow model being executed in different ways. In addition, the concepts that were described by the WFMC captured only simple control-flow dependencies. This meant that workflow migration was not only hampered by different interpretations of similarly named concepts, but also by the fact that some concepts were supported in one environment but did not have a counterpart in another.

These were the circumstances that gave rise to the Workflow Patterns Initiative in the second half of 1999. The original founders recognized that there was a need to distill the essential features of the many workflow management systems that existed. This would allow an unbiased comparison of different approaches to the specification of executable business processes and provide the basis for the adaptation and refinement of existing techniques as well as supporting the development of new approaches.

The approach chosen focussed on identifying the constructs required for the specification of control-flow dependencies between tasks. The ability to explicitly capture tasks and their chronological dependencies is an essential capability of workflow management systems. Initially, 13 commercial workflow management systems and two research prototypes were examined with respect to the constructs they offered for control-flow specification. This resulted in a collection of 20 control-flow patterns. Following the book by Gamma et al., which provided patterns for object-oriented design, patterns have become a popular way of identifying recurring problems and corresponding solutions in various areas of computer science. The Workflow Patterns consisted of a description of desired control-flow functionality, problems with realizing this functionality and implementation approaches. Workflow management systems were rated on a three point scale against these patterns; the highest score was given where there was direct support for a pattern, while the intermediate score indicated there were some restrictions in terms of direct pattern support. The lowest score did not mean that the pattern could not be realized from a theoretical point of view, as scripting languages are all Turing-complete, but simply that the system involved did not provide direct support. Therefore, the patterns are not concerned with expressive power but with suitability, a notion that refers to the alignment between a modeling language and a problem domain.

Over time, the patterns-based evaluations were extended to business process modeling languages (e.g., UML Activity Diagrams, BPMN), standards for web
service composition (e.g., BPML, BPEL), and open-source workflow management systems (e.g., jBPM, OpenWFE). The patterns collection itself was revised and significantly extended with data patterns, resource patterns, exception handling patterns, service interaction patterns, flexibility patterns, etc.

1.5 Petri Nets and Workflow Nets

While there has not been a commonly accepted formal foundation for workflow management, in the mid nineties, Wil van der Aalst articulated three reasons why Petri nets would make a good candidate. The theory of Petri nets, developed by Carl Adam Petri, provides an elegant approach to describing and solving concurrency related problems.

A Petri net is a bipartite graph where the nodes are either places or transitions. A Petri net has a graphical representation where places are represented by circles, transitions by squares, and their connections by directed arcs. A Petri net may have an associated marking, which is an assignment of tokens to places. A marking represents the state of a system. A transition is said to be enabled in a certain marking when each of its input places contains at least one token. An enabled transition can fire by taking a token from each of its input places and producing a token in each of its output places. Transitions thus correspond to allowed state changes. A vast body of theory exists for the formal analysis of Petri nets.

As van der Aalst pointed out, the graphical nature of Petri nets, their explicit representation of the notion of state, and the existence of analysis techniques made them eminently suitable for workflow specification. To increase their suitability for workflow specification, he introduced workflow nets. A workflow net is a Petri net with one start place and one end place. All tasks have to be on a path from the start place to the end place. To simplify the representation of typical workflow routing constructs, such as AND-splits and XOR-splits, a number of graphical abbreviations were introduced. Workflow nets form a subclass of Petri nets for which the analysis of desirable properties, for example, whether process instances of a workflow can always terminate, is feasible. Tool support for this type of analysis was developed in the form of tools such as Woflan (Workflow Analyzer), ProM, WoPeD, etc.

1.6 The Emergence of YAWL

After the development of the initial collection of workflow patterns, Petri nets were revisited in terms of their suitability for the specification of control-flow dependencies in workflows. While it turned out that many of the patterns could be expressed in a straightforward manner, some patterns were not so easily captured. This observation led to the development of YAWL (Yet Another Workflow Language). In YAWL, Petri nets were taken as a starting point 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.

YAWL was given a formal semantics based on a state transition system and therefore its nets cannot be simply mapped to Petri nets. Hence, while YAWL is inspired by Petri nets, it cannot be seen as a set of notational abbreviations on top of this formalism. Specific verification approaches needed to be developed to deal with the new constructs offered by YAWL. One formalism that turned out to be particularly useful for reasoning about YAWL nets were reset nets. These are Petri nets extended with the concept of a reset arc. When a transition is executed, all tokens are removed from places that are connected with a reset arc to this transition. Reset arcs allow the concept of cancelation to be directly expressed. In workflows cancelation implies that the execution of a certain task should lead to other nominated tasks being terminated or not being available for further execution. YAWL offers direct support for cancelation, while its predecessor, workflow nets, does not.

After YAWL was defined, work started on the implementation of a support environment. This effort intended to demonstrate that it was possible to provide comprehensive support for the (original) workflow control-flow patterns. As such the YAWL environment can be seen as a reference implementation that supports the Workflow Patterns. The term YAWL became synonymous with both the language and the support environment. Over time as the environment evolved, the ambitions increased and YAWL grew into a full-fledged workflow management system, which is used in a wide variety of academic and industrial settings.

1.7 A Brief Overview of YAWL

To illustrate and further concretize some of the concepts we have mentioned thus far, let us consider a simplified scenario involving a car accident. In this scenario, the first step that needs to be taken is to obtain a quote for the costs involved in dealing with the damage. After this quote has been received, a preliminary insurance claim is lodged, and a choice needs to be made whether the car is going to be fixed or whether buying a new car is more cost-effective. After this latter decision has been made the bill can be settled, and when this has happened and the preliminary insurance claim has been lodged, the final insurance claim can be lodged.

In Fig. 1.3, a Petri net is shown that captures the flow of control of this simple example. The rectangles in this net are the transitions, which correspond to tasks that need to be performed, while the places correspond to moments in-between processing. The place that is input for the transitions Have Car Fixed and Buy New Car captures a choice made by the environment. When a token has been produced by the transition Obtain Quote for this place, both transitions Have Car Fixed and Buy New Car are enabled. Once one of these has been chosen, the token is removed from this place and the other transition is no longer enabled. When a transition has multiple output places, this represents parallelism as all these places receive a token upon
Fig. 1.3  A Petri net for the Accident workflow

Fig. 1.4  A YAWL net for the Accident workflow

completion of this transition. Similarly, if a transition has multiple input places, this represents a synchronization point as at least one token is required in each input place for this transition to be enabled.

Figure 1.4 shows the YAWL representation of the Petri net of Fig. 1.3. As can be seen, in a YAWL net, there is a more explicit representation of the join and split behavior of the various tasks. In addition, there is no need to explicitly represent all places; those that simply connect two tasks can be omitted. The YAWL model captures the order in which the tasks need to be presented at runtime. To make the specification available to the runtime environment, it needs to be specified in the Editor, see Fig. 1.5, which can save the model in an XML format that can be interpreted by the Engine. In addition, one needs to specify how work is assigned to participants. To keep this example simple, let us assume that all tasks are to be executed by the Claimant, which we have captured as a role with one participant. Also, tasks are offered to the Claimant and he/she can choose when to start working on them.

At runtime, the Engine uses the YAWL model to determine when certain tasks are to be offered to the Claimant. In Fig. 1.6, one can see the worklist containing the work items (i.e. task instances) offered to the Claimant after the task Obtain Quote has been performed. Note that both options Buy New Car and Have Car Fixed are offered and the Claimant needs to make a choice.

Assuming that the Claimant completed the choice for having their car fixed or having the current one repaired, the task Settle Bill is offered. In Fig. 1.7, the data input form for the corresponding work item is shown. One can see that performing this work item involves filling in a number of data fields. The presentation of these fields is governed by the type of the data involved (which in this example are all simple types) and whether the fields contain editable values or are for presentation
purposes only. Upon completion of the work item, this information is sent back to the Engine, which, in general, may pass it on to other tasks or use it to determine which tasks should be performed next.
The simple example illustrates that in the YAWL environment, as in virtually all BPM environments, there is a distinction between build time and runtime. Models are constructed in the build time component, the Editor (discussed in detail in Chap. 8), and subsequently deployed in the runtime environment. The runtime environment itself can consist of a number of components, but at its core are the Engine (discussed in detail in Chap. 9) and the Resource Service (discussed in detail in Chap. 10). The Engine deals with the control-flow logic and workflow data, while the Resource Service is concerned with the routing of work items to appropriate resources. A simplified overview of the YAWL architecture is presented in Fig. 1.8 (a detailed discussion of the YAWL architecture can be found in Chap. 7).

1.8 Positioning of YAWL

YAWL has a number of features that position it uniquely in the crowded field of BPM. The language could be developed without the pressures of vested interests and a sole focus on providing comprehensive support for the Workflow Patterns was possible. In contrast to BPMN, comprehensive control-flow patterns support was achieved through the introduction of a minimal set of constructs, rather than a construct-per-pattern approach. More recently, comprehensive support for the resource patterns was realized in YAWL, both the language and the support environment.

Another distinguishing feature of YAWL is the fact that it has a formal foundation, that is, both a precisely defined syntax and a precisely defined semantics. In the latest version of YAWL, the (abstract) syntax has been defined through the use of set
theory and predicate logic, while the semantics has been defined in terms of a large Colored Petri net (CPN), which can interpret YAWL specifications. This formal foundation removes any ambiguity associated with the interpretation of complex constructs and their interplay and also allows for the development of sophisticated verification techniques that allow the detection of inherent flaws in an executable process model before it is deployed (for a detailed treatment of verification in YAWL see Chap. 20). While it is sometimes claimed that certain standards or oft-used approaches have a formal foundation, the problem is usually that either (1) the connection between the language and the formal theory remains unclear, or (2) the formalization is not generally accepted, and certainly not by a standard body.

The importance of sophisticated flexibility support in BPM systems has long been recognized. Because of the complexity of providing this support, it has taken quite a long time for satisfactory solutions to become available. Flexibility requirements may take different forms. For example, exceptions may arise that were not anticipated in advance, processes may be more easily captured by rules rather than through an explicit representation of all possible execution paths, and processes may
evolve over time due to changes in a business and/or its environment. As in different contexts different solutions may be desirable, YAWL not only provides services supporting various flexibility requirements but also allows these services to inter-operate. This leads to a powerful approach in dealing with such requirements. For example, it is possible to combine the procedural YAWL language with worklets (cf. Chap. 4) and Declare (cf. Chap. 6). Worklets can be used to select process fragments at runtime based on rules. Declare can be used to enact processes based on (temporal) constraints.

The YAWL support environment is open source and is therefore not only freely available but can also be extended as desired, thus avoiding vendor lock-in. Its service-oriented architecture, with a rich set of interfaces, allows the system to interact with other systems and to be extended in a variety of ways.

Through a link with the ProM environment, YAWL logs can be analyzed in a number of ways and YAWL specifications can be simulated. For example, it is possible to do a simulation that starts with the current state and that looks into the near future, for example, to decide whether it is beneficial to hire more resources or not. The log analysis can reveal, among others, what the typical walk-throughs are through a process or what the probabilities are of the process choosing one alternative path over another.

Finally, one can leverage the benefits of BPMN through the BPMN2YAWL plug-in, where BPMN models can be mapped onto YAWL specifications. Contrary to BPEL, the resulting YAWL models remain readable and maintain a close link with the original BPMN models. This is a consequence of the fact that in YAWL arbitrary cycles can be specified and that YAWL offers comprehensive support for the control-flow patterns in general. The BPMN2YAWL plug-in is discussed in detail in Chap. 13.

1.9 Overview of the Book

This book is about YAWL, both the language and the open source support environment. It discusses the foundations of the language, its unique support for flexibility, the support environment, services that can interoperate with this environment, its relationship to other well-established approaches in the field of BPM, and several applications. There are many technical papers that focus on various aspects of YAWL. This book aims to make this work accessible to a wider audience and is therefore less technical in nature. Where possible, references are provided so that the interested reader can deepen her understanding of specific topics. Another objective of this book is to bring the main material concerning YAWL together in one place and to properly integrate this material. To facilitate the understanding of the reader, a running example is provided in the area of supply chain management.

The book is divided into nine parts. Part II provides the conceptual foundation of YAWL. It explains the concepts required for workflow specification and introduces the running example.
Workflows may need to evolve over time, have to deal with unforeseen exceptions, and sometimes their specification is more easily achieved by specifying constraints that need to be satisfied at runtime rather than by providing an explicit road map for all possible execution paths. In Part III, various aspects of flexibility are examined.

Part IV discusses the core YAWL environment, its architecture, its design component, and its main runtime component. YAWL’s architecture is service-oriented, which means that its various components communicate through well-defined interfaces. The design component allows the creation and verification of workflows. Once completed, they can be passed to the runtime component, the Engine, for execution.

In Part V, a number of services are presented, which can interact with the core YAWL environment and provide additional functionality. The services discussed provide concrete ways of achieving flexibility in workflow specification and execution (the Worklet Service and the Declare Service) or deal with work distribution, allowing the Engine to remain resource-agnostic (the Resource Service).

Part VI positions YAWL with respect to a number of well-known approaches to business process specification and/or execution, the Business Process Modeling Notation (BPMN), Event-driven Process Chains (EPCs), the Business Process Execution Language (BPEL), and a number of other well-known open-source workflow management systems. It is shown how BPMN models can be mapped to YAWL nets, how EPCs compare to YAWL and how they can be mapped to YAWL nets and vice versa, how BPEL relates to YAWL, and how open-source workflow management systems such as jBPM and OpenWFE compare with YAWL.

In Part VII a number of advanced topics in the field of BPM are examined. The connection with the Process Mining (ProM) framework is discussed and it is demonstrated how this framework can be used to mine valuable information from execution logs that have been generated by the YAWL environment. Correctness notions for YAWL specifications are introduced and precisely defined and it is shown how, and to what extent, these notions can be automatically verified. Sometimes new process models can be derived from existing process models and the notion of process configuration is explored in more depth through the C-YAWL (Configurable YAWL) approach. Finally, workflows may be running in different settings (e.g. different organizations or departments), but there may be a need for them to exchange information. Ideally such needs are expressed at the specification level, not at the implementation level.

Part VIII discusses two applications in which the YAWL environment has been used. One of these applications is in the healthcare domain, and the other is in the domain of film production.

Part IX concludes the book and provides an outlook for future developments.
Exercises

**Exercise 1.** Why is it important that a workflow specification language has a formal syntax and semantics? Provide at least three reasons.

**Exercise 2.** Explain the difference between the notions of expressive power and suitability in the context of a workflow specification language.

**Exercise 3.** Explain the value proposition of patterns.

Chapter Notes

The original founders of the Workflow Patterns Initiative were Wil van der Aalst, Alistair Barros, Arthur ter Hofstede, and Bartek Kiepuszewski. The first paper appeared in the CoopIS conference in Eilat in 2000 [7] and the main reference appeared in the Distributed and Parallel Databases journal in 2003 [16]. Since then there have been many additional contributors. Since 2005 Nick Russell has been the main driver in developing workflow patterns for the data [225] and the resource perspectives [222] as well as for exception handling [221]. He also led the revision effort of the original control-flow patterns [224]. Marlon Dumas and Petia Wohed have been involved in many patterns-based evaluations (see, e.g., [84, 223, 262–265]). Nataliya Mulyar devoted her PhD work [178] to identifying additional patterns. The main reference site for the Workflow Patterns is www.workflowpatterns.com. A book on the Workflow Patterns is expected to be published by MIT Press in the near future.

An early influential book in the area of workflow management was the book by Stefan Jablonski and Christoph Bussler [124]. Among others, a number of perspectives were described for workflow management, some of which are referred to in this chapter, though under different names. The book by Wil van der Aalst and Kees van Hee [11] is a more recent book on workflow management and, among other topics, discusses in depth the application of Petri nets in this area. Mathias Weske’s book [260] discusses the original control-flow patterns, but also workflow nets and YAWL, and, more generally, provides a treatment to the field of business process management from a computer science perspective. Although it is impossible to provide an overview of workflow literature here, we would also like to mention the workflow books by Leymann and Roller [150], Marinescu [157], zur Muehlen [174], and Lawrence [146]. A recent overview of important topics can be found in the book on process-aware information systems [82].

Patterns development became popular in computer science due to the work of Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [100] in the area of software design patterns. Other examples of well-known patterns collections in software analysis and design were developed by Gregor Hohpe and Bobby Woolf [120] and by Martin Fowler [98].
Kiepuszewski et al. [131] examined the consequences of various different interpretations of workflow routing constructs defined by the Workflow Management Coalition from an expressiveness point of view.

There are several comprehensive treatments of Petri nets (e.g., [179, 196]). Their use for workflow specification was advocated by Wil van der Aalst [1]. Dufourd et al. [80, 81] provide an introduction to reset nets.

In 2002, van der Aalst and ter Hofstede defined the YAWL language [15]. The development of the support environment started the following year [6]. The first open-source release of this environment occurred in the same year. In subsequent releases, the functionality was extended considerably. In [219], YAWL and some of its extensions were modeled using CPN Tools.
Modern Business Process Automation
YAWL and its Support Environment
ter Hofstede, A.H.M.; van der Aalst, W.; Adams, M.; Russell, N. (Eds.)
2010, XVIII, 676 p., Hardcover
ISBN: 978-3-642-03120-5