II BPM FUNDAMENTALS

2 Business Process Modeling

In this chapter, we introduce the area of business process modeling, the practice of capturing the details associated with the operation of business processes in written form. In doing so, we define the fundamental concepts that underpin the act of business process modeling, examine the objectives and history of the field, and outline popular contemporary modeling techniques.

2.1 Fundamental concepts

The genesis of the business process modeling field lies in the areas of *process modeling*, which seeks to describe the dynamic aspects of processes and has its antecedents in the area of mathematics, and *system modeling*, which stems from the need to describe software systems and their intended operation. *Business process modeling* seeks to provide a holistic description of an operational process and considers not only the issues of task coordination and control-flow but also data definition and usage in the context of the process and the organizational environment (organizational structure, resources, reporting lines, etc.) in which it will operate.

A *process* is considered to be a collection of related activities together with associated execution constraints that lead to some anticipated end goal or outcome. Although the notion of a process has general applicability, it is the execution of a process in the context of a given organizational setting — commonly termed a *business process* — that is of most interest for us in this book.

A business process is described by a *business process model*, which defines its means of operation. This may be at an *abstract* level, where there is no consideration of how it will ultimately be effected and the focus is on the type of work activities making up a process and the sequence in which they are undertaken, or at a *concrete* level, where it is described in sufficient detail such that the model can be directly executed by a software system.

Figure 2.1 outlines the various aspects associated with business process design and execution. A dedicated software system called the *business process designer* or *editor* is used to design the business process model. Once this contains sufficient detail, it can be utilized to guide the operation of the *Business Process Management System (BPMS).* This involves loading the business process model into the BPMS and executing it in conjunction with any other information that is required to facilitate its operation. This may include data associated with resources and the organizational structure from an *organizational repository* (e.g., HR systems, LDAP, or X.500 databases), data from *external databases*, and the results of interactions with *external systems* and *web services*. The interaction of *process*

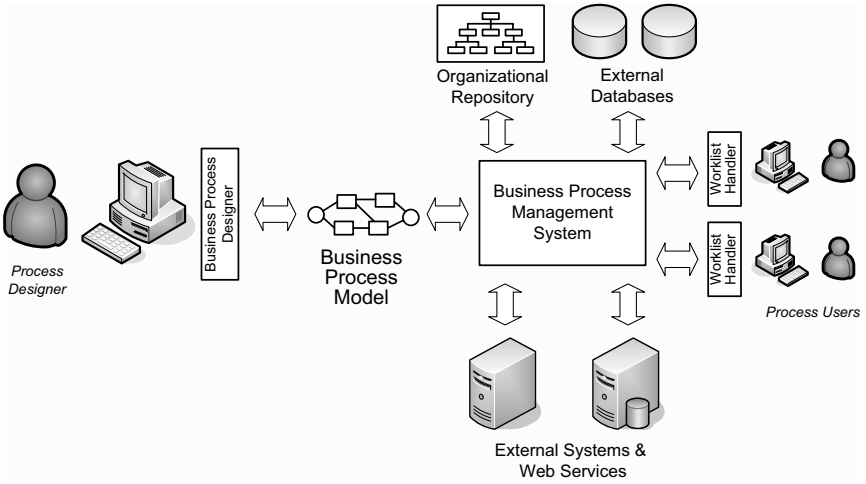


Figure 2.1
Design and execution of a business process.

participants with the business process occurs via individual *worklist handlers*, which display the work allocated to specific users and allow them to signal how they intend to deal with it.

The business process model is typically centered around a description of the control-flow aspects of the process in the form of a directed graph, in which the constituent *tasks*, which describe the individual work activities that comprise the process, are connected by directed arcs indicating the various execution paths through the process. Different forms of gateway constructs (splits and joins) allow alternate execution paths to be taken through the process model. Business process models can be *hierarchical* in nature as the operation of an individual task within a model can be described in terms of a more detailed process specification, a concept commonly known as *decomposition*.

Figure 2.2 outlines the main control-flow and data concepts relating to a business process model in the form of a UML class diagram. A *specification* defines a particular process model within an overall business process. A specification is composed of *tasks*, *gateways*, and *arcs*. As shown by the *contains* relationship, a task appears in precisely one specification. Although not shown, this also holds (implicitly) for gateways and arcs: they are only added to describe the routing of tasks within a specification. Gateways define the various splits and joins that connect (and diverge) branches within process models. Within a given process specification, directed arcs connect tasks and/or gateway nodes and define the execution sequence within the process model. Each arc has precisely one source and one target (task or gateway). In some cases (e.g., the individual output branches from an XOR-split), the triggering of an individual arc is based on the positive evaluation of an

associated Boolean *condition*. Similarly, specifications and tasks may have *preconditions* and *postconditions* associated with their execution defining the time at which they are able to be enabled and can complete, respectively.

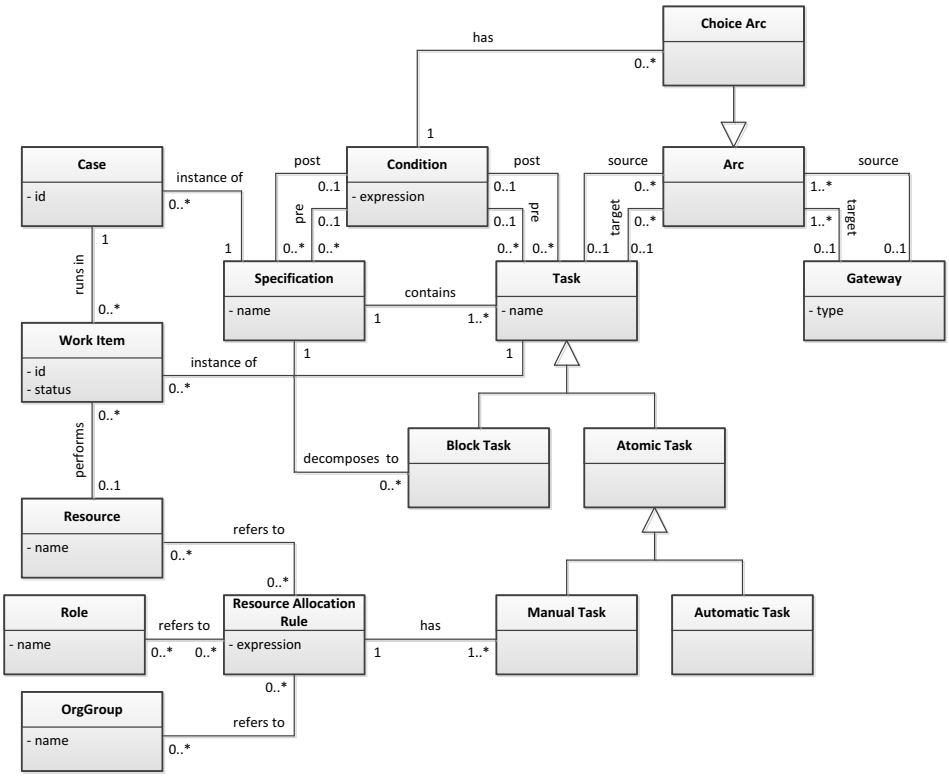


Figure 2.2

UML class diagram of main control-flow and resource concepts.

A *task* corresponds to a single unit of work. An *atomic task* has a simple, self-contained definition (i.e., one that is not described in terms of other tasks). A *block task* (also known as a *composite task)* is a complex activity whose implementation is described in terms of a *subprocess*, which has an associated process specification. As shown by the *decomposes to* relation in figure 2.2, a block task corresponds to precisely one specification, but multiple block tasks may refer to the same specification. When a block task is started, it passes control to the first task(s) in its corresponding subprocess. This subprocess executes to completion, and at its conclusion it passes control back to the composite task.

Tasks can be of two types: *manual* or *automatic.* A *manual task* is allocated to one or more resources (usually human resources) for completion, whereas an *automatic task* can be undertaken without requiring any human input (e.g., by executing a program or invoking

a web service). A manual task has an associated *resource allocation rule* that indicates how the corresponding work items will be allocated to resources. The resource allocation rule is an expression that may refer to individual *resources*, *roles*, and *organizational groups*. An example expression is "Task A should be executed by a manager (role) in the sales department (organizational group) or by Pete (resource)." An *organizational group* is formal grouping of resources within an organization, and a role is a set of resources with a common characteristic.

Each invocation of a task is termed a *work item*. Specific invocations are undertaken by at most one resource. Each invocation of a complete business process model is termed a *case*. Each work item executes within the context of a particular case. Usually there is one work item initiated for each task in a given case; however, where a task forms part of a loop, a distinct work item is created for each iteration of the loop (e.g., the task is invoked each time the thread of control cycles around the loop). Similarly multiple instances of a task are created for a designated "multiple instance" task construct when it is initiated.

There may be multiple cases of a particular process running simultaneously; however, each of these is assumed to have an independent existence, and they typically execute without reference to each other. There is usually a unique first node and a unique final node in a process. These are the starting and finishing points for each given process instance. The execution of a case is typically signified by the notion of a *thread of control* (of which there may be several in any given process instance), which indicates the current execution point(s) within the process. The initiation of a case occurs when a new thread of control is placed in the starting node for a process. The execution of the case involves the traversal of the thread of control from the start node to the end node. As a consequence of the presence of split gateway operators in a process, it is possible for the thread of control to be split into several distinct threads of control during execution of a process instance. Similarly, join gateway operators in a process provide a mechanism for execution threads on several incoming branches to be merged into one execution thread on a single outgoing branch.

A desirable quality of a process, however, is that it is *sound* as this excludes some of the basic (but relatively commonplace) process design flaws. Soundness is defined with respect to a start node (initial state) and an end node (desired end state). A process is sound if (1) it is always possible to reach the end state (there is always an option to complete, i.e., no deadlocks or livelocks), (2) upon reaching the end state there should be no other threads of control, and (3) for every task there should be at least one scenario from start to end where it is executed [1, 11].

Just as soundness is one possible property of a process based on its structural characteristics, there are other properties that a process may demonstrate. *Structuredness* refers to a structural property of a process where splits and joins are balanced as illustrated in figures 2.3(a) and (d). In figures 2.3(b) and (c), there are examples of processes that seem to be structured but are not. In figure 2.3(b), the split and join types don't match, and in figure

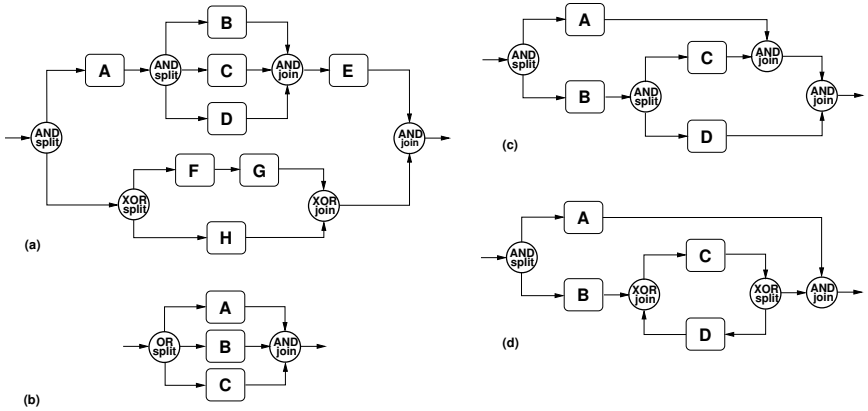


Figure 2.3
The concept of structuredness: illustrative examples.

2.3(c), the splits and joins are balanced but overlap. All of the processes are sound except for that in figure 2.3(b).

One of the desirable characteristics of a structured process model is that it is also *safe*. In an unsafe model, there may be multiple indistinguishable threads of control belonging to the same case concurrently visiting a node in the process. As a result, there may be two indistinguishable task instances.

Traditionally, much of the focus of business process modeling has been on the control-flow aspects; however, there are also other considerations in regard to data and resource handling that are of particular importance when attempting to develop business process definitions that closely mirror real-life scenarios. The representation and usage of data within a process is of fundamental importance as in practice, almost every process involves the coordination of business tasks based on various items of information available both within and outside of the context of the business process. We consider a distinction between two types of data associated with a process model: (1) *model data*, which refer to data elements used in the structural definition of the business process model — these are essentially static in nature for a given process model and do not change once a business process has been defined (i.e., their values are identical for each instance of the process), and (2) *production data*, which relate to actual working data elements that are directly utilized and may change during the operation of the business process (i.e., they are dynamic in nature).

The set of *production data* associated with a given process instance defines its *state* at any given point in time. This takes the form of a series of data elements, which may be simple or structured in form and have a value (or set of values) associated with them. Data elements may be used for a variety of purposes during process execution, including

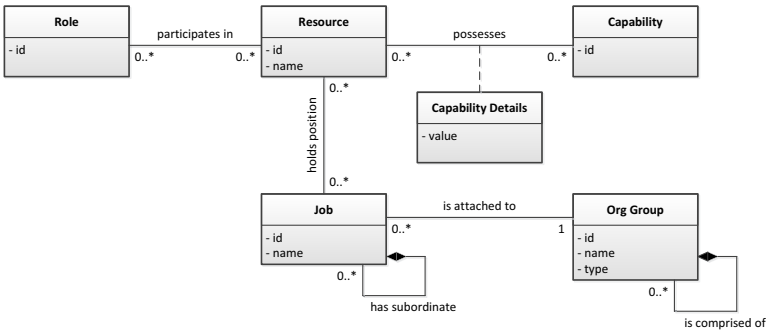


Figure 2.4
UML class diagram of main organizational concepts.

communication of data between task instances and process instances, routing decisions when deciding which branches to take during execution, and when to start and complete a task instance (*preconditions* and *postconditions*).

Each data element has a specific *scope* that defines the region in a given process in which it is available. This may be as broad as every instance of the process, throughout an entire case associated with the process, or as limited as a given task instance. The scope of a data element also determines its lifespan. Case data elements may exist for the entire time that a case executes, whereas a data element whose scope is limited to a specific task may only exist for a given execution instance of the task within a case. Should the task execute more than once, then there may be more than one instance of the data element during the execution of the case. For a given case, the notion of *state* refers to the complete collection of model and production data elements. This collection completely specifies the current execution status of the case.

The other aspect of a business process that merits further discussion relates to *resources* (i.e., the entities that actually undertake the work items in the business process). In general, it is assumed that a business process is defined in the context of an organization and utilizes specific resources within the organization to complete the tasks of which the business process is composed. Hence (other than for fully automated processes), there is usually an *organizational model* associated with each business process that describes the overall structure of the organization in which it operates and captures details of the various resources and the relationships between them that are relevant to the conduct of the business process. Figure 2.4 outlines the main organizational concepts relating to business processes in the form of a UML class diagram.

A resource is considered to be an entity that is capable of doing work. Although resources can be either human or non-human (e.g., plant and equipment), we will focus on human resources. Work is assigned to a resource in the form of work items, and they receive

notification of work items distributed to them via a *worklist handler*, a software application that provides them with a view of their work items and supports their interaction with the system managing overall process execution. Depending on the sophistication of this application, users may have one or several work queues assigned to them through which work items are distributed. They may also be provided with a range of facilities for managing work items assigned to them through to completion.

A human resource is assumed to be a member of an *organization*. An organization is a formal grouping of resources that undertake work items pertaining to a common set of business objectives. Resources usually have a specific *job* within that organization, and in general, most organizational characteristics that they possess relate to the position(s) that they occupy rather than directly to the resource themselves. There are two sets of characteristics that are exceptions to this rule, however: *roles* and *capabilities*.

Roles serve as another grouping mechanism for human resources with similar job roles or responsibility levels (e.g., managers, union delegates, etc.). Each resource may have one or more corresponding roles. Individual resources may also possess *capabilities* or attributes that further clarify their suitability for various kinds of work. These may include qualifications and skills as well as other job-related or personal attributes such as specific responsibilities held or previous work experience.

Each *job* is attached to an *organizational group*, which is a permanent group of human resources within the organization that undertake work items relating to a common set of business objectives. Organizational groups may be comprised of other (subordinate) organizational groups and thus form a hierarchy within the organization. Within the organizational hierarchy, each job may have a number of specific *subordinates* for whom they are responsible and to whom each of them reports.

Of particular interest from a resource perspective is the manner in which work items are distributed and ultimately bound to specific resources for execution. Figure 2.5 illustrates the lifecycle of a work item in the form of a state transition diagram from the time that a work item is created through to final completion or failure. It can be seen that there are a series of potential states that comprise this process.

Initially, a work item comes into existence in the *created* state. This indicates that the preconditions required for its enablement have been satisfied, and it is capable of being executed. At this point, however, the work item has not been offered or allocated to a resource for execution, and there are a number of possible paths through these states that the individual work item may take.

Transitions from the *created* state are typically initiated by the system. They center on the activity of making resources aware of work items that require execution. This may occur in one of three distinct ways denoted by the subsequent states. A work item may be *offered to a single resource*, meaning that the system informs exactly one resource about the availability of a work item. It may do this by sending a message to the resource or adding the work item to the list of available work items that the resource can view. Inherent in this

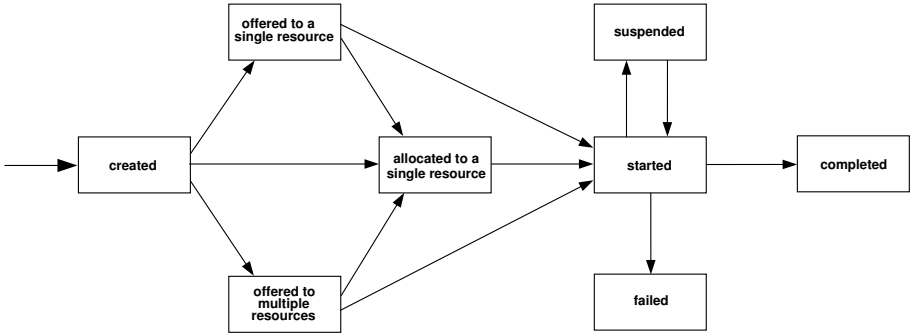


Figure 2.5
Basic work item lifecycle (from [113]).

is the notion of the system selecting a specific resource to which the work item should be advertised. This may occur in a variety of different ways — the process model may include specific directives about the identity of the resource to which a given work item should be directed or it may be based on more general requirements such as utilizing the least busy, cheapest, or most appropriately qualified resource. In each of these situations, there is the need to determine which resources are suitable and available to undertake the work item and then to rank them and select the most appropriate one.

An alternative to this course of action is indicated by the state *offered to multiple resources*, where the system informs multiple resources of the existence of a work item. Again the notion of resource selection applies; however, in this case, the system informs all suitable resources of the work item. It does not attempt to identify which of them should undertake it. This is now left to the resources (i.e., they can "compete" for work or divide the work in an informal manner) (outside of the system).

The *allocated to a single resource* state denotes the state of a work item where a specific resource has committed to its execution at some time in the future. A work item may progress to this state either because the system pre-emptively allocates newly created work items to a resource or because a resource volunteers to undertake a work item that has been offered.

Subsequent states in the work distribution model are *started*, which indicates that a resource has commenced executing the work item, *suspended*, which denotes that the resource has elected to cease execution of the work item for a period but does intend to continue working on it at a later time, *failed*, which identifies that the work item cannot be completed and that the resource will not work on it any further, and *completed*, which identifies a work item that has been successfully executed to completion.

2.2 Objectives of business process modeling

There are a variety of reasons for undertaking business process modeling, and in many cases, the desired outcome is more than just a series of diagrams capturing a business process. In this section, we discuss the motivations for business process modeling in more detail.

2.2.1 Business process definition

One of the major benefits offered by the act of business process modeling is that it provides a guided approach to gaining an understanding of the most important aspects of a business process and documenting them in a form that alleviates any potential for ambiguous interpretation. The notion of modeling the dynamic aspects of processes is not particularly novel and has been undertaken in the software design field for many years; however, it is only with the advent of richer, multiperspective business process modeling techniques that the accurate capture of the broader range of aspects relevant to business processes has been possible in a single model.

2.2.2 Business process enactment

Although in some cases business process modeling may be an objective in its own right, more generally, the development of such models is an intermediate step to their ultimate automation using some form of BPMS. As such, the business process model serves as a design blueprint for the subsequent software development and deployment activity. It may also serve as a means of identifying the most appropriate enactment technology.

2.2.3 Business process communication

The use of business process models provides an effective means of communicating their intention and operation in an unambiguous way to the various parties involved in their operation. As the underlying technology matures and the ambition level associated with business process automation initiatives increases, this becomes increasingly important as often the parties involved are not located within a single department or at a single operating location within a single organization but increasingly are located across a variety of different geographic locations and may even be members of different organizations (e.g., upstream suppliers or downstream customers). Business processes increasingly span organizational boundaries, and cross-organizational business process automation offers opportunities for enhancing and optimizing these processes in ways that were previously not possible. As the various parties to such a process will frequently utilize differing enabling technologies,

business process models provide a way of communicating the operational expectations associated with these processes in an organizationally and technologically independent way.

2.2.4 Business process analysis

Business process models provide a good starting point for all kinds of analyses. For example, there are striking similarities between simulation models and the models used to enact processes using a WFM or BPM system. In the context of business process modeling, the two main types of analysis are *verification* and *performance analysis*.

The goal of verification is to find errors in systems or procedures (potential deadlocks, livelocks, etc.) in an early stage (i.e., before enactment). *Model checking* can be used to verify that the modeled system exhibits a set of desirable properties. Unfortunately, despite the availability of powerful analysis techniques and tools, few organizations apply verification techniques. This is unfortunate because empirical research shows that modelers tend to make many errors. For example, in [81], it is shown that more than 20 percent of the process models in the SAP reference model contain serious flaws (deadlocks, livelocks, etc.). Such models are not aligned with reality and introduce errors when used to configure the system. Unreliable and low-quality models provide little value for their end users.

Performance analysis aims to analyze processes with respect to key performance indicators such as throughput, response times, waiting times, and utilization. Techniques like *simulation* can be used to better understand the factors influencing these performance indicators Moreover, "what-if" scenarios can be analyzed and compared. This helps managers to improve and manage processes.

Verification and performance analysis are typically model-driven (i.e., the modeled reality is analyzed rather than reality itself). However, information systems are becoming more and more intertwined with the operational processes they support. As a result, a multitude of events are recorded by today's information systems. Reality can be observed by inspecting such event data. The goal of *process mining* is to use this data to extract process-related information (e.g., to automatically *discover* a process model by observing events recorded by some enterprise system) [3]. By linking business process models and historic and current data in the information system, new types of analyses come into reach. For example, process mining can reveal factual bottlenecks and quantify conformance.

2.2.5 Business process compliance

Many organizations have deployed automated business processes in recent years; however, changing business conditions and the relative inflexibility of the enabling technologies often means that the actual business process enacted by members of an organization on a day-to-day basis does not necessarily match that described in the original or intended business process model. Nonetheless, many corporate quality and risk management systems

assume adherence to a given set of operating norms, which are often based on process models. Indeed, recent legislation such as the Sarbanes-Oxley Act [118] requires that U.S. organizations demonstrate compliance with a set of standard business processes in order to mitigate potential business risk.

Although country-specific, there is a large degree of commonality among Sarbanes-Oxley (US), Basel II/III (EU), J-SOX (Japan), C-SOX (Canada), 8th EU Directive (EURO-SOX), BilMoG (Germany), MiFID (EU), Law 262/05 (Italy), Code Lippens (Belgium), Code Tabaksblat (The Netherlands), and others [3]. These regulations require companies to identify the financial and operational risks inherent to their business processes and establish the appropriate controls to address them. Although the focus of these regulations is on financial aspects, they illustrate the desire to make processes transparent and auditable. The ISO 9000 family of standards is another illustration of this trend. For instance, ISO 9001:2008 requires organizations to model their operational processes. Currently, these standards do not force organizations to check conformance at the event level. For example, the real business process may be very different from the modeled business process.

An increasing area of focus for organizations deploying automated process solutions is the comparison of recorded execution history with standard corporate processes to assess the degree of compliance. To ensure that this activity is effective, precise business process models are a necessity. *Conformance checking* techniques [3, 5, 39] compare modeled behavior with observed behavior. These techniques basically try to replay reality on the modeled process. If there are deviations, then these are recorded, and the most similar path through the model is chosen.

Whereas conformance checking techniques compare observed event sequences with execution paths of the model, other business process compliance approaches compare different models (where models can also be just rules). For example, in [59], process models are compared with contracts. In other approaches, the specification model is compared with the implementation model (e.g., using refinement or inheritance notions) [7].

2.3 History of process modeling

The notion of process modeling has existed in various forms for several decades, and its specific application to business processes has been a particular objective for much of this time. The timeline shown in Figure 2.6 illustrates various techniques that have achieved some degree of popular use over the past 70 years and their relationship with other similarly founded techniques. As part of this analysis, we also give a relative evaluation of the *expressiveness* of each technique, by which we refer to the ability of the technique to capture the various aspects associated with a business process in a precise and holistic manner. These techniques fall into three main groups: software design methods, formal techniques, and dedicated BPM techniques. We discuss each in turn.

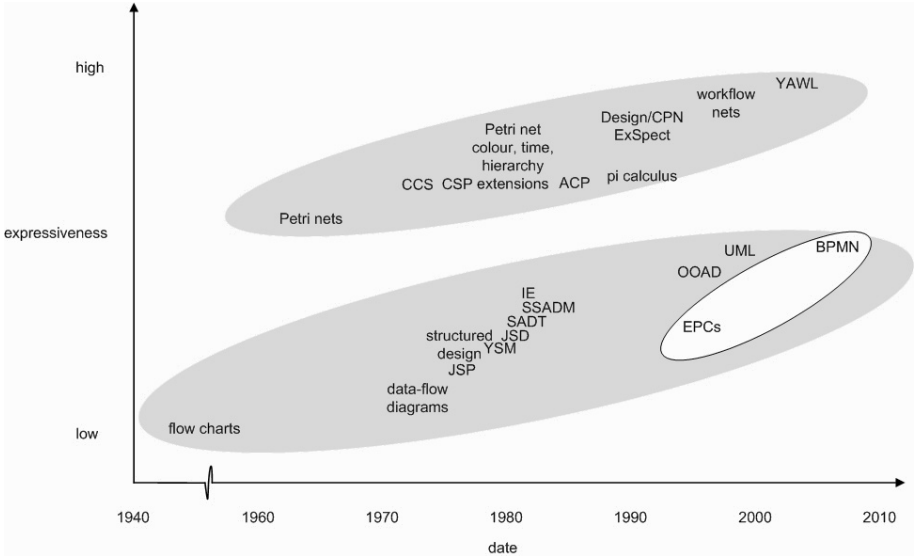


Figure 2.6

Timeline of process modeling techniques. Three main groups can be identified: software design methods (e.g., UML), formal techniques (e.g., Petri nets), and dedicated BPM techniques (e.g., BPMN).

2.3.1 Software design methods

Software design methods stem from the field of computing and aim to provide mechanisms for describing the dynamic aspects of processes. Much of the focus of these methods is on providing design-time support for the capture of processes with the intention of subsequently automating them using various forms of computing technology. With the advent of mechanized process support in the 1940s, it was soon apparent that a means of describing the structure of and execution flow through computer programs was required. Although there is some debate as to who was responsible for recognizing their applicability, it is clear that the flow process chart pioneered by Gilbreth in 1921 was introduced to the computing field in the period 1944–1947. Commonly known as flowcharts, the technique provides a graphical means of representing the set of steps in a process or algorithm as boxes connected by lines denoting the flow of control between them together with decision nodes identifying points in the process whether control-flow might take alternate paths on the basis of a specific decision.

Perhaps the next significant advance in the development of BPM techniques was the development of the Data Flow Diagram (DFD) in 1974, where a software system was viewed as a sequence of data transformations and individual programs effectively operated as functions on the data elements that flowed between them. The implicit notions behind

DFDs — that program units should be based on groups of tightly related functions, that the dependencies between them should be minimized, and that their coupling should be precisely defined — quickly led to the establishment of the structured design field for defining software systems and spawned techniques such as Jackson Structured Programming (JSP) as a means of describing software programs.

The initial successes associated with the structured design field subsequently led to the establishment of the notion of structured analysis, which proposed that an entire software system should be viewed in terms of the data flowing through it. It introduced additional modeling notations such as structure charts and context diagrams to show the manner in which program functionality was decomposed into individual functions and how they were assigned to available processing resources. In addition to the original proposal, this design philosophy also led to a plethora of related techniques, including Jackson Structured Design (JSD), Yourdon Structured Method (YSM), Structured Analysis and Design Technique (SADT), Structured System Analysis and Design Method (SSADM), and Information Engineering (IE).

The next significant advance in the software design field was the rise of Object-Oriented Analysis and Design (OOAD) methodologies, where the design of a software system was conceptualized in terms of tightly bound functional units known as classes and the interaction between individual instances of these classes known as objects. The major proponents of these techniques included Booch, Rumbaugh, and Jacobson, who devised their own OOAD technique, complete with various new ways of describing both the static and dynamic aspects of system operation. In 1997, the most significant aspects of the three techniques were merged into the Unified Modeling Language (UML), a methodology for describing software systems comprising 13 distinct modeling notations. Of these, the activity diagram notation is the most useful for modeling business processes.

2.3.2 Formal techniques

One of the criticisms associated with the use of many of the software design techniques described in the previous section is that they are informally defined; as a consequence, there is the potential for ambiguities and inconsistencies to arise where they are used for modeling business processes. To remedy this, a number of formal process modeling techniques have been developed in recent years, which have a fully defined operational semantics ensuring there is no potential for ambiguity to arise in their usage.

Perhaps the most well-known formal technique for process modeling is that of Petri nets devised by Carl Adam Petri in 1962. Petri nets have been widely used in a number of domains for modeling and analyzing concurrent systems and, more recently, for capturing dynamic aspects of business processes. One of the drawbacks with the original definition of Petri nets was that it lacked data concepts requiring that any notions of data manipulation be explicitly represented in the model. Another difficulty stemmed from the absence of the

notion of hierarchy, allowing a model to be neatly decomposed into submodels with defined interfaces. There was also no notion of time in the original modeling formalism. High-level Petri nets, developed during the 1970s, added the notions of color (data support), hierarchy, and time to the original definition and addressed these deficiencies. The precise operational foundation associated with Petri nets and their graphical nature makes them an effective technique for modeling and evaluating process models, and several design and simulation tools have been developed for both basic Petri nets and variants of them. Two widely used tools supporting modeling and simulation of Colored Petri nets are Design/Colored Petri Nets (CPN) (1989) and ExSpect (early 1990s). Later Design/CPN was replaced by CPN Tools, providing enhanced support for simulation, animation, and analysis of models. Petri nets triggered the development of workflow nets — a variant of standard Petri nets suited to the specification of workflow processes — and also motivated the creation of YAWL (Yet Another Workflow Language), a multiperspective workflow modeling language and reference workflow management system implementation.

Process algebra techniques provide another stream of formal methods that have proven useful for modeling dynamic processes. Although they lack the intuitive graphical form of Petri nets, process algebras offer a powerful technique for modeling and analyzing concurrent systems and more recently have proven useful in capturing mobile processes. Significant process algebra initiatives include Calculus of Communicating Systems (CCS) during the period from 1973 to 1980, Communicating Sequential Processes (CSP) developed in 1978, and the Algebra of Communicating Processes (ACP) established in 1982. Most recently, the π-calculus, a successor to CCS, has been advocated as a suitable basis for modeling business processes, particularly mobile processes, which involve changing communication links between active process elements during execution.

2.3.3 Dedicated BPM techniques

The increasing interest in BPM in recent years and the growing use of business process automation tools such as workflow systems have led to several modeling techniques specifically developed for capturing business processes. In the early 1990s, Event-Driven Process Chains (EPCs) were developed at the University of Saarland as a means of modeling business processes in terms of functions and events. EPCs have been widely used as the modeling basis in a number of commercial tools such as the popular ARIS Toolset.

In 2003, the Business Process Modeling Initiative (BPMI), an industry consortium, proposed the Business Process Modeling Notation (BPMN), the first attempt at a standard for capturing business processes. In its original form, BPMN was a graphical representation for specifying business processes and was intended as a notation capable of specifying complex processes in a form that is intuitive to both technical and business users. The BPMN standard proposed a mapping to the WS-BPEL execution language for enactment purposes, although this was not complete. The latest release of BPMN (version 2.0.2, now

titled Business Process Model and Notation) extends the original modeling notation with a more comprehensive underlying metamodel, which enables all aspects of a business process to be specified. Where there is sufficient detail captured for a given model, the BPMN metamodel provides the ability for it to be directly executed. A number of current BPMS offerings, including the Oracle BPM Suite examined in this book, provide for the direct execution of a BPMN process model without the need for the process designer to map it to an intermediate execution model such as WS-BPEL, as has previously been the case. The BPMN standard is currently managed under the auspices of OMG.

2.4 Overview of popular techniques

As can be seen from the preceding section, over time, a variety of techniques have been proposed for business process modeling. Many of these did not achieve any degree of prominence or are no longer in popular usage. However, the increased focus on the area in recent years has seen a convergence on five specific approaches for modeling business processes. In this section, we introduce each of these techniques and discuss their capabilities and shortcomings.

2.4.1 Petri nets

Petri nets were developed by Carl Adam Petri as part of his doctoral thesis in 1962 [102]. He proposed them as a means of describing the operation of discrete distributed systems. Petri nets (which are also known as place/transition nets [P/T] nets) are a particularly useful means of describing the dynamics of processes. One of their attractions is that they have a simple graphical format together with precise operational semantics that provide an intuitive way of modeling both the static and dynamic aspects of processes.

Figure 2.7 illustrates the language constructs for Petri nets. A Petri net takes the form of a directed bipartite graph where the nodes are either *transitions* or *places*. Places represent intermediate stages that exist during the operation of a process, and transitions correspond to the activities or events of which the process is made up. *Arcs* connect places and transitions in such a way that places can only be connected to transitions and vice versa. It is not permissible for places to be connected to places or for transitions to be connected to other transitions (i.e., places and transitions alternate along any path in the graph). A directed arc from a place to a transition indicates an input place to a transition, and a directed arc from a transition to a place indicates an output place from a transition. These places play an important role in the overall operational semantics of Petri nets.

The operational semantics of a Petri net are described in terms of *tokens*, which signify a thread of control flowing through a process. Places in a Petri net can contain any number of tokens. The distribution of tokens across all of the places in a net is called a *marking*

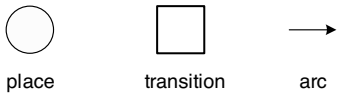


Figure 2.7
Petri net constructs (from [64]).

and signifies the overall state of the Petri net. A transition in a Petri net can "fire" whenever there is one or more tokens in each of its input places. When it fires, it consumes a token from each input place and adds a token to each output place. Figures 2.8(a) – (e) illustrate the act of firing for various Petri net configurations. It is assumed that the firing of a transition is an atomic action that occurs instantaneously and cannot be interrupted.

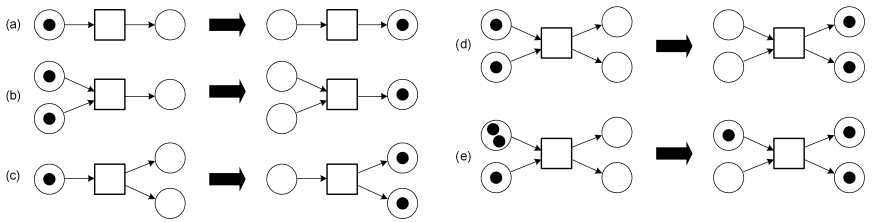


Figure 2.8
Petri net operational semantics (from [64]).

The manner in which a Petri net executes is deliberately intended to be non-deterministic. Hence, when multiple transitions in a Petri net are enabled, any one of them may fire, although it is not possible for two transitions to fire simultaneously as firing is considered atomic (assuming interleaving semantics). Furthermore, a distinction is drawn between a transition being able to fire and it actually firing, and one of the salient features of Petri nets is that an enabled transition is not obliged to fire immediately but can do so at a time of its choosing. These features make Petri nets particularly suitable for modeling concurrent process executions such as those that occur in business processes.

Figure 2.9 depicts a licence application process in the form of Petri net. In this process, an applicant lodges an *application* for a new license, which then triggers two distinct lines of activity: (1) a testing sequence, which consists first of a *written test* and then a *practical test* within a given timeframe otherwise a *timeout* occurs, and (2) determination of whether any *special requirements* apply to the license. If they do, then there is an iterative process to *investigate* each requirement and *assess* whether it should apply to the license. When the testing and special requirements sequences have been completed, a *recommendation* is made regarding the application, and on the basis of a *positive decision*, a *pass* action occurs for the license or, alternatively, on the basis of a *negative decision*, a *fail* action

occurs. Regardless of the outcome of the application, at the conclusion of each case, the application is subject to *archive*.

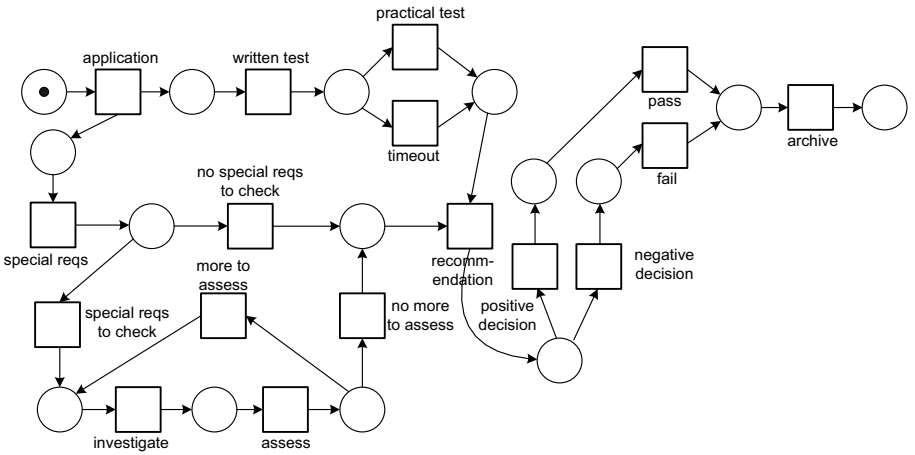


Figure 2.9
License application process: Petri net.

Despite its relative simplicity, this process model gives an insight into the use of Petri nets for modeling business processes. As a means of comparison with other modeling techniques, the same process model will be used for illustrative purposes throughout this chapter.

2.4.2 Workflow nets

Workflow nets were developed by Wil van der Aalst as a means of describing workflow processes in a precise way, making them amenable to subsequent analysis at both design-time and runtime [1]. Workflow nets are based on Petri nets with some additional constraints:

1. A workflow net has a single start place and a single end place; and
2. Every transition in the workflow net is on a path from the start place to the end place.

It is also desirable that a workflow net is sound; hence, a *sound workflow net* has the additional property that for any case, the procedure can always eventually terminate. At the moment of termination, there will be precisely one token in the end place, and all other places in the workflow net will be empty. Moreover, there should be no dead transitions — for any arbitrary transition in the net, there should be a scenario in which this transition can be executed by following an appropriate route through the net.

The format of workflow nets is essentially the same as for Petri nets, although some specific operators encompass the particular sorts of operations that arise in workflow systems. These operators can be seen as “syntactic sugar” and provide a shorthand notation

for common constructs. Figure 2.10 illustrates the range of workflow net constructs. Transitions as they arise in Petri nets are replaced with the task construct; however, the essential structural form of a workflow net is largely the same as for a Petri net.

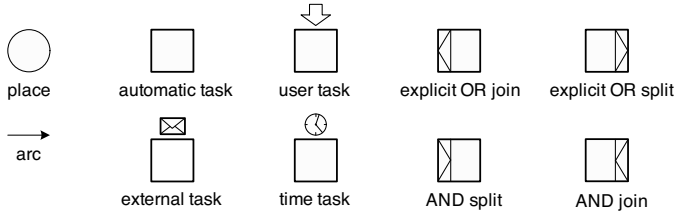


Figure 2.10
Workflow net constructs (from [64]).

Tasks can be initiated in one of four ways: (1) *automatic* tasks execute as soon as they are enabled, (2) *user* tasks are passed to human resources for execution once enabled, (3) *external* tasks only proceed once they are enabled and a required message or signal is received from the operating environment, and (4) *time* tasks only proceed once they are enabled and a specified (time-based) deadline occurs. It is also possible for split and join behavior to be associated with individual tasks.

As an illustration of the form that a workflow net takes, figure 2.11 shows the earlier license application example in the form of a workflow net. It has a similar format to the Petri net shown in figure 2.9. However, AND/XOR splits/joins are represented explicitly, and the diagram also shows the way in which tasks are initiated (i.e., either by users or on the basis of a time-based trigger).

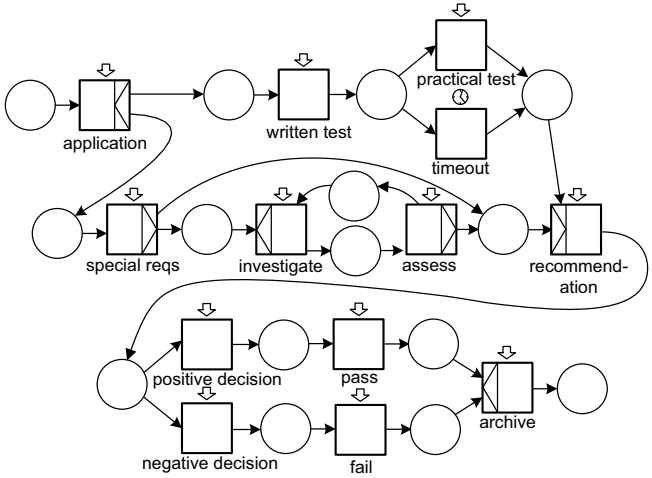


Figure 2.11
License application process: workflow net.

2.4.3 EPCs

Event-driven process chains (EPCs) were developed by August-Wilhelm Scheer as part of the ARIS framework at the University of Saarland in Germany in the early 1990s. Figure 2.12 illustrates the main EPC constructs. Essentially, an EPC model describes a business process in terms of a series of function-based transformations between a set of input events and a set of output events. An EPC model takes the form of a directed graph, which always starts and ends with events. The set of constructs that make up an EPC model is shown in figure 2.12. The AND, OR, and XOR constructs can be utilized as either join or split operators.

Figure 2.12
EPC constructs.

Within the model, events correspond to potential states that may exist within a process, and functions essentially describe actions that are undertaken to transform one state (or set of states) to another. Hence, a function is always preceded and followed by an event. It is not permissible for an event to be linked either directly (or indirectly through a join or split construct) to another event, nor for a function to be linked to another function (i.e., events and functions alternate along any path in the graph). Where several events precede or follow a function (i.e., events and functions alternate along any path in a model), one of the AND, OR, or XOR constructs can be used to define the intended split or join behavior between the event and function nodes. Figure 2.13 illustrates the range of allowable and invalid split and join configurations between events and functions. It is important to note that the use of OR-split and XOR-split constructs immediately after an event (or directly preceding a function) is not allowed under the EPC modeling formalism.

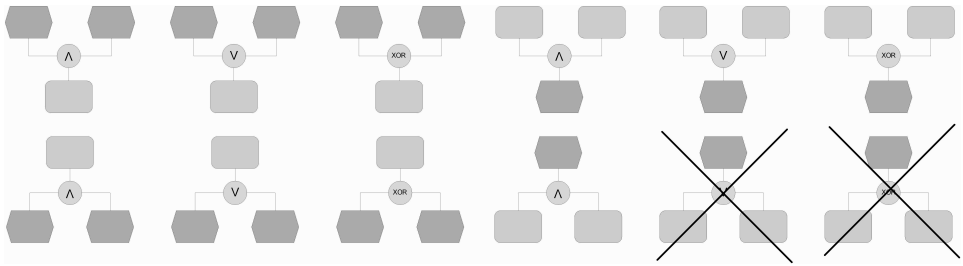


Figure 2.13
EPCs: Allowable and invalid join and split combinations.

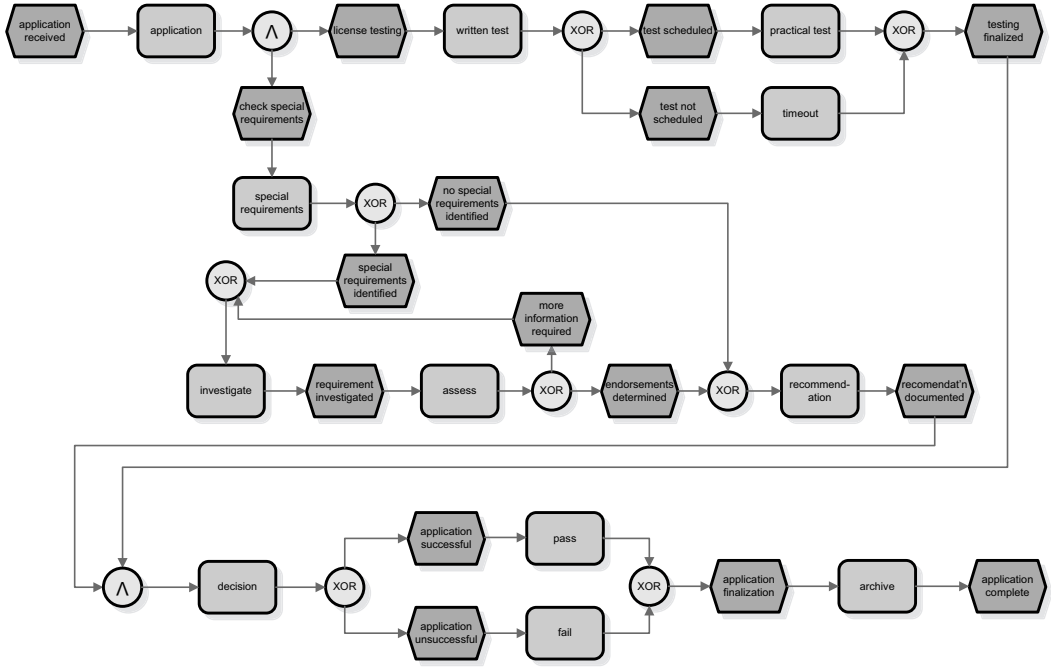


Figure 2.14
License application process: Event-driven process chain.

Figure 2.14 shows an example of an event-driven process chain. In this case, it depicts the license application process, which has also been shown earlier as a Petri net and a workflow net. This figure mirrors the process depicted in the Petri net and workflow net models in figures 2.9 and 2.11 with one minor exception: EPCs lack the ability to model a situation where a user- or environment-initiated action makes an explicit choice as to the course of action taken at a specific decision point in a process. In the Petri net and workflow net models, the ability to specify that the *practical test* task has a *timeout* associated with it can be directly captured as both modeling formalisms incorporate an explicit notion of state; however, in the EPC model, an explicit choice is required as to whether the *practical test* or *timeout* task should execute.

This decision can only be made at the expiry of the timespan for the *practical test* task and in effect is made by the system based on state information available to it rather than as a result of a specific user intervention. Hence, the EPC process model is subtly different from the earlier models, although in most practical situations the outcome will ultimately be the same.

2.4.4 BPMN

Business Process Model and Notation (BPMN) is an XML-based language for defining business processes with a formal meta-model and an associated graphical notation. In its current incarnation (version 2.0.2), it is designed to be utilized at one of three levels:

- as a high-level, graphical, business modeling language (with a limited range of modeling constructs);
- as a detailed graphical modeling language (with a comprehensive range of modeling constructs); and
- as an executable business process language, capturing sufficient details about the control-flow, data, and resource aspects of a business process that it can be directly executed by a business process engine.

The initial version of BPMN (then termed Business Process Modeling Notation) was proposed by the Business Process Modeling Institute (BPMI), an industry consortium, in 2002 as a graphical modeling language. It was informally defined and intended primarily for high-level modeling of business processes.

Since 2005, it has been under the auspices of the Object Management Group (OMG), when it merged with BPMI. Its focus has now broadened from business process modeling to offering a range of techniques supporting all aspects of business process capture, from high-level, abstract modeling through to detailed business process definition capable of direct execution. All of these techniques co-exist within the same meta-model, thus ensuring that the semantics of the various levels are consistent.

For this discussion, we will focus on the graphical modeling notation as it is the aspect of the BPMN that is most widely used for high-level modeling of business processes. As with other business process modeling notations, it depicts a process in the form of a directed graph that connects a series of *flow objects* (essentially events, activities, joins, and splits) and *data objects* (which identify data resources and the way in which they are utilized in the process) using various types of *connecting objects* (arcs that identify different types of object flowing through the process). *Swimlanes* are used to organize the various parts of the process into parts, which are undertaken by the same class of resource. *Artifacts* allow additional details to be added to the process. Figure 2.15 illustrates the main graphical constructs in BPMN 2.0.2. These divide into five main categories.[1]

1 Note that when referring to BPMN artifacts in the text, they are identified as proper nouns and hence capitalized, e.g., Task, Pool, etc.

Flow objects

Flow objects describe the main active elements in a process: in particular, *events* that signal that nominated start, intermediate, and end parts of the process have been reached, *activities* that correspond to the actual conduct of work in a process, and *gateways* that indicate the various types of split and join constructs within a process.

Data

Data denote individual information objects and collections that exist within the context of a process. *Data objects* identify data elements that are consumed and produced by activities within the process. *Data inputs* and *data outputs* identify formal parameters to processes and subprocesses. *Data stores* denote permanent repositories of data that exist beyond the life of individual process instances.

Connecting objects

Connecting objects describe the various flows within a process. *Sequence* flow connects activities and indicates the order in which they will be done. *Message* flow depicts the flow of messages between two participants in a process. It typically connects two separate pools. One end connects to a pool boundary and the other to a flow object within a pool boundary. The direction of the message flow indicates the initiating and receiving participant for the message. *Conditional* flow corresponds to a sequence flow between activities that has a condition associated with it, which is evaluated at runtime to determine whether the flow will be used (i.e., whether control-flow will pass along it). *Default* flow identifies a path from a decision construct (i.e., a *data-based exclusive gateway* or *inclusive gateway*) that is selected if all other conditional flows are not chosen (because conditions associated with them evaluate to false). *Associations* are used to link information with flow objects. *Data Associations* identify mappings between data objects and an activity or event in a process. The data association may indicate the direction of data flow or it may be unidirectional in the situation where it is linked to a sequence flow (in which case it assumes the direction of the sequence of the sequence flow). *Exception flow* occurs outside normal flow and identifies how an activity deals with a particular type of event that arises during its execution. *Compensation Association* also occurs outside normal flow and indicates how the failure of a transaction or the occurrence of a compensation event within an activity are dealt with and compensated for.

Swimlanes

Swimlanes provide a means of grouping elements within a process. A *pool* identifies a participant in a collaboration and provides a means of partitioning specific activities from

other pools. Pools may be *white-box*, in which case they contain flow elements and are labeled with the process to which they correspond, or they may be *black-box*, in which case they have no internal details and are labeled with the role or business entity to which they correspond. A *lane* is a subpartition within a pool that is used to categorize and organize activities within it. They can be used for a variety of types of categorization (e.g., to indicate activities undertaken by a specific actor, role, or organizational group or even to partition activities on a specific basis, e.g., chargeable vs non-chargeable).

Artifacts

Artifacts provide additional information about a process. Two distinct types of artifact are denoted: *groups* provide a visual mechanism for grouping related activities, and *text annotations* provide a means of adding additional information to a process model.

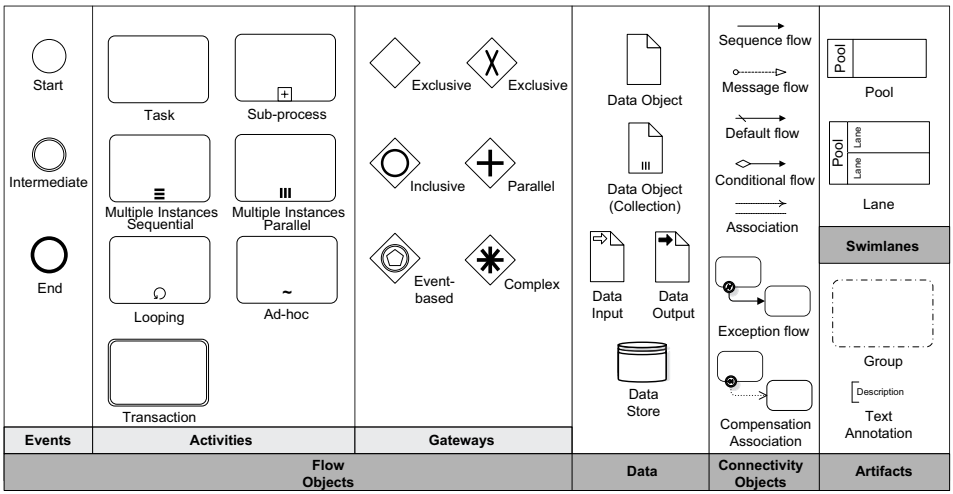


Figure 2.15
BPMN constructs.

Event handling

An important aspect of BPMN is that it provides support for modeling exception flow as part of the business process. This is done using events that can be included in the sequence flow within a process or can be attached to either individual activities or subprocesses. Each event is of a specific type, indicating what form of exception is being handled. The actual definition of the handling strategy for a given event can be specified in one of two ways: (1) a path from the event construct to the specific sequence of activities that can deal with the event is specified and the thread of control is passed to this branch when the event arises, or (2) if there is not a specific handling strategy specified, the thread of control is

instead passed to an appropriate handler in the surrounding block (or if this is already the outermost block, then the process instance is terminated).

For events that form part of sequence flow, they may either be *catching* events, meaning that they are triggered when the execution thread has arrived at the event and a specific type of trigger signal is received, or they may be *throwing* events, which generate a specific type of signal when the execution thread reaches the throwing event during normal execution.

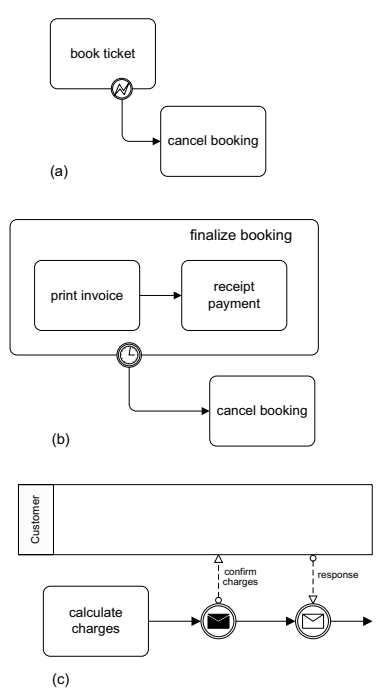
Events that are attached to the boundary of an activity or a subprocess denote boundary events that are listened for during the execution of the activity (or subprocess) and where the nominated trigger signal is received cause the exception flow from the event to be triggered. There are two types of boundary event distinguished by different event notations: *interrupting*, which causes the execution of activity to cease on receipt of the trigger signal and the exception flow to be triggered, and *non-interrupting*, which immediately trigger the exception flow on receipt of the trigger signal but also allow the activity to continue and for normal sequence flow to be triggered from the activity when it completes.

Figure 2.16(a) shows how an exception handler can be specified to handle errors encountered in the *book ticket* activity. In figure 2.16(b), a timer event is attached to the *finalize booking* subprocess, allowing it to be terminated if it is not completed by a specified deadline. Figure 2.16(c) illustrates the use of inline throwing and catching events. In this case, an interaction occurs with an external *Customer* participant involving message events to confirm acceptance of charges. Figure 2.16(d) illustrates the range of events that are recognized within BPMN. Event detection and handling can be associated with start, intermediate, and end activities depending on whether the event is signaled from outside the process or detected inside during its operation and whether it can be handled within the process during the course of its operation, such that execution can continue or whether it results in the process instance terminating.

As a comparative illustration, figure 2.17 shows the license application process in the form of a BPMN model. Of particular interest are (1) the specific exception construct associated with the *practical test* activity to handle the timeout if it is not completed within a required timeframe, and (2) the relatively complex gateway structure required to handle the optional *investigate – assess* task cycle, which comprises part of the *special requirements* branch within the process.

2.4.5 UML activity diagrams

UML activity diagrams are part of the UML modeling framework that provide a means of describing the operation of a process. Although originally intended for describing software processes, they have also shown themselves to be useful for describing business processes. Like BPMN, UML is managed under the auspices of OMG. The current version of the UML standard (2.5) was finalized by the OMG in 2015. Figure 2.18 gives an overview of the main UML constructs. These are divided into four groups.



(d)

Event Type	Catching		Throwing		Non-Interrupting	
	Start	Intermediate	Intermediate	End	Start	Intermediate
Message	✉	✉	✉	✉	✉	✉
Timer	🕒	🕒			🕒	🕒
Error	⚡	⚡		⚡		
Escalation	▲	▲	▲	▲	▲	▲
Cancel		✕		✕		
Compensation	◀◀	◀◀	◀◀	◀◀		
Conditional	≡	≡			≡	≡
Link		⇨	➡			
Signal	△	△	▲	▲	△	△
Terminate				●		
Multiple	⬠	⬠	⬟	⬟	⬠	⬠
Parallel Multiple	✚	✚			✚	✚

Figure 2.16
Event handling in BPMN.

Actions

An *Action* corresponds to a single step within an activity in a business process. It can be either atomic and self-contained in terms of its definition or in the case of a *Call Behavior Action*, it may involve a call to another activity. An *Action* can have preconditions and postconditions associated with its execution, which can be graphically depicted. The *AcceptEventAction* waits for the occurrence of an event meeting a specified condition. This event may be triggered from within the process instance via a *SendSignalAction* or it may be a time-based event (in which case the receiving *AcceptEventAction* is denoted by a different symbol).

Nodes

Nodes are constructs that interlink actions in a process and further clarify the specific details associated with control and data flow. An *InitialNode* marks the starting point for a process. A process may be terminated by either an *ActivityFinal* node, which immediately causes the process instance to terminate when the first execution thread reaches it, or a

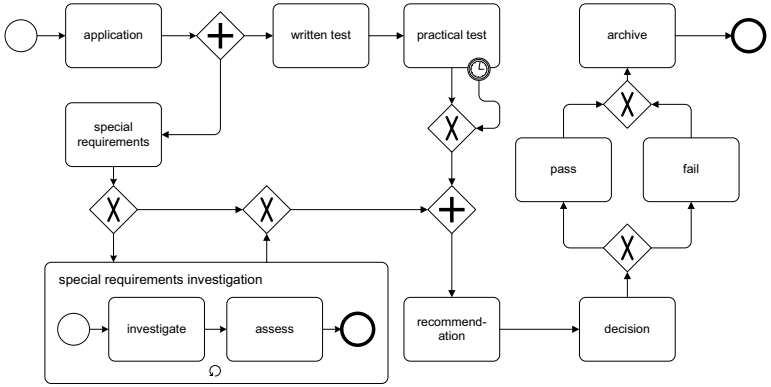


Figure 2.17
License application process: BPMN.

FlowFinal node, which simply consumes execution threads but allows the process instance to continue until all active execution threads have reached an end node. A *DecisionNode* allows the thread of control in the incoming branch to be split into one or more outgoing branches on a conditional basis, and the corresponding *MergeNode* allows threads of control in multiple incoming branches to be merged into a single outgoing branch on an unconditional basis. The *ForkNode* provides the ability to split the thread of control in an incoming branch into several outgoing branches, and the corresponding *JoinNode* supports the synchronization of the execution threads in multiple incoming branches, only allowing

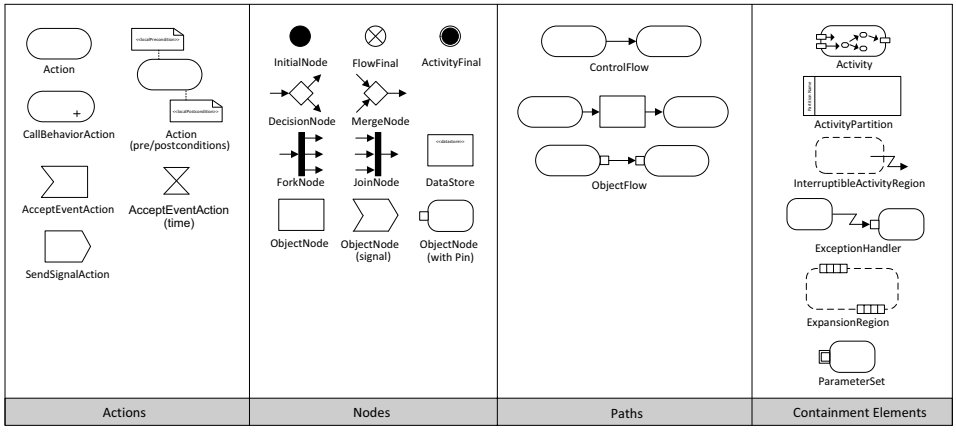


Figure 2.18
UML 2.5 activity diagram constructs.

the outgoing branch to be triggered when all incoming branches have been triggered. The *DataStore* node denotes a persistent store of data within a process. Three types of *ObjectNode* identify abstract activity nodes that are part of the object flow in an activity. These may correspond to data elements and signals used within a process and may include *Pins*, which denote input and output data values.

Paths

There are two types of paths identifying flows within a UML 2.5 activity diagram: *ControlFlow* identifies how the thread of control is passed between nodes in a process, and *ObjectFlow* signifies the flow of objects, data elements, tokens and signals in a process.

Containment elements

Containment elements provide various mechanisms for grouping nodes within a process model for specific purposes. *Activity* elements serve as a means of grouping related individual actions into a single step in a process. There is provision for passing parameters to and from individual activities. An *ActivityPartition* provides a means of grouping actions that are performed by the same organizational enitity. An *InterruptibleActivityRegion* is a means of grouping related activity nodes that have the same exception handling behavior. The various steps that make up a given approach to handling a specific exception are characterized by an *ExceptionHandler* node. An *ExpansionRegion* is a structured activity region that executes multiple times on the basis of the parameters passed to it. A *ParameterSet* denotes a complete set of inputs or outputs to an *activity*.

Figure 2.19 illustrates the use of UML 2.5 activity diagrams for modeling business processes using the license application example discussed earlier.

2.5 Analysis of process models

Process models can be used to analyze the correctness and performance of business processes. Moreover, as more and more event data is stored in enterprise information systems, processes can be analyzed easily using factual data. First, we summarize two mainstream approaches for model-based analysis: *verification* and *performance analysis*. Verification is concerned with the correctness of a system or process. Performance analysis focuses on flow times, waiting times, utilization, and service levels. After discussing these two types of model-based analysis, we focus on techniques that exploit event data when analyzing processes and their models.

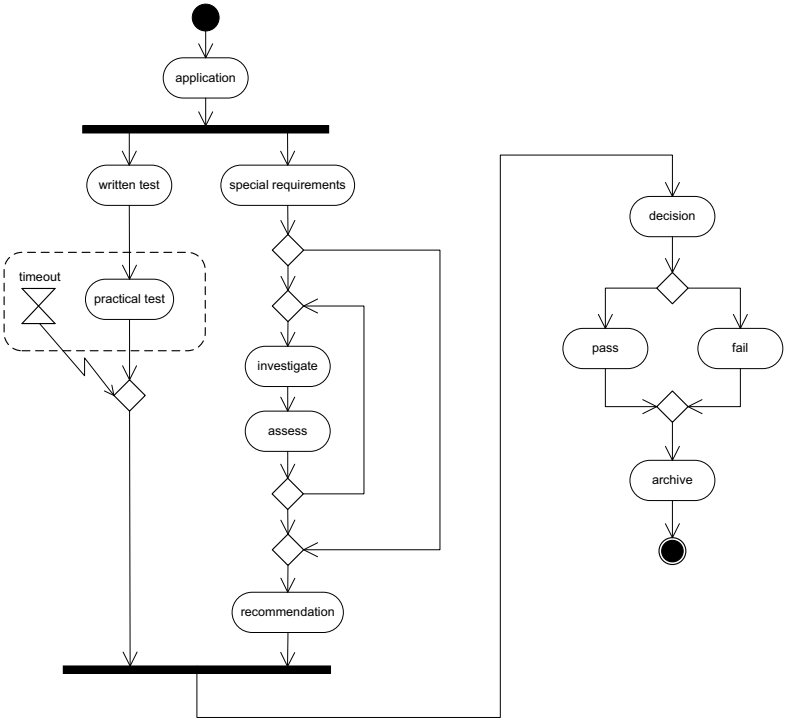


Figure 2.19
License application process: UML 2.5 activity diagram.

2.5.1 Verification

When modeling a process, it is easy to make design errors. Consider, for example, the process shown in figure 2.20. The model will always deadlock just before reaching task *archive*. This tasks wants to synchronize two flows, but only one of them will be activated after task *recommendation*.

The workflow net shown in figure 2.20 is *not sound*. As described earlier, a workflow net is sound if and only if (a) "proper completion" is guaranteed (i.e., when the unique end place is reached all other places should be empty), (b) the process always has the "option to complete" (i.e., no matter what path is chosen, it is always possible to reach the end of the process), and the workflow net contains no "dead parts" (i.e., in principle all tasks are reachable) [1]. The workflow net of figure 2.20 does not have the "option to complete" as it gets stuck before task *archive*. The notion of soundness can easily be adapted for other languages such as YAWL, EPCs, and BPMN.

Soundness is a generic property. Sometimes a more specific property needs to be investigated (e.g., "there should be a practical test for all failed applications"). Such properties

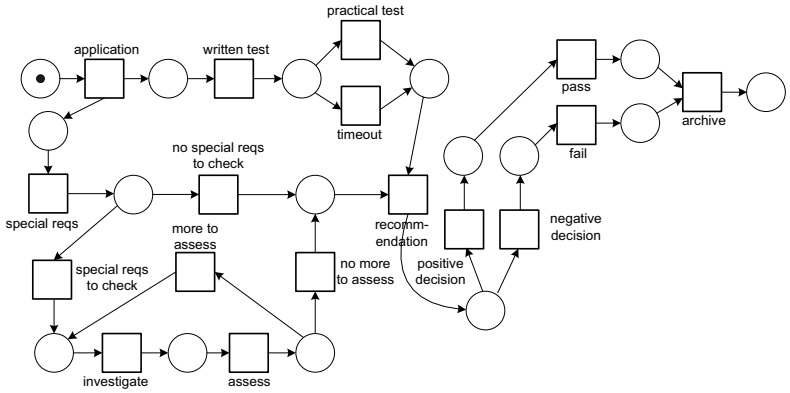


Figure 2.20
A workflow net that is not sound.

can be expressed in *temporal logic* [34]. *Linear Temporal Logic* (LTL) is an example of a temporal logic, which in addition to classical logical operators uses temporal operators such as always ($\Box$), eventually ($\$\Diamond\$$), until ($\sqcup$), weak until (W), and next time ($\bigcirc$). The expression $(\Diamond x) \implies (\Diamond y)$ means that for all cases where x is executed, y is also executed. Another example is $\Box(x \implies \Diamond y)$, which states that any occurrence of x will be followed by y. *Model checking* techniques can be used to check such properties [34].

Another verification task is related to the comparison of two models. For example, the implementation of a process needs to be compared to the high-level specification of the process. Different equivalence notions exist (e.g., trace equivalence and branching bisimilarity) that can be checked automatically.

There are various tools to verify process models. A classic example is Woflan, which is tailored toward checking soundness [127]. Also workflow systems such as YAWL provide verification capabilities [64].

2.5.2 Performance analysis

The performance of a process or organization can be defined in different ways. Typically, three dimensions of performance are identified: *time*, *cost*, and *quality*. For each of these performance dimensions, different *key performance indicators* can be defined. When looking at the *time dimension*, the following performance indicators can be distinguished:

- The *lead time* (also referred to as flow time) is the total time from the creation of the case to the completion of the case. In terms of a workflow net, this is the time it takes to go from the start place to the end place. One can measure the average lead time over all cases. However, the degree of variance may also be important (i.e., it makes a difference

whether all cases take more or less two weeks of if some take just a few hours while others take more than one month). The *service level* is the percentage of cases having a lead time lower than some threshold value (e.g., the percentage of cases handled within two weeks).

- The *service time* is the time actually worked on a case. One can measure the service time per activity (e.g., the average time needed to make a decision is 35 minutes) or for the entire case. Note that in case of concurrency, the overall service time (i.e., summing up the time spent on the various activities) may be more than the lead time. However, typically, the service time is just a fraction of the lead time (minutes versus weeks).
- The *waiting time* is the time a case is waiting for a resource to become available. This time can be measured per activity or for the case as a whole. An example is the waiting time for a customer wanting to talk to a sales representative. Another example is the time a patient needs to wait before getting a knee operation. Again one can be interested in the average or variance of waiting times. It is also possible to focus on a service level (e.g., the percentage of patients that has a knee operation within three weeks after the initial diagnosis).
- The *synchronization time* is the time an activity is not yet fully enabled and waiting for an external trigger or another parallel branch. Unlike waiting time, the activity is not fully enabled yet (i.e., the case is waiting for synchronization rather than a resource).

Performance indicators can also be defined for the *cost dimension*. Different costing models can be used (e.g., Activity Based Costing (ABC), Resource Consumption Accounting (RCA), etc). The cost of executing an activity may be fixed or depend on the type of resource used, its utilization, and/or the duration of the activity. Cost may depend on the utilization of resources. A key performance indicator in most processes is the *average utilization* of resources over a given period (e.g., an operating room in a hospital has been used 85% of the time over the last two months). A detailed discussion of the various costing models is outside of the scope of this book. The *quality dimension* typically focuses on the "product" or "service" delivered to the customer. Like cost, this can be measured in different ways. One example is customer satisfaction measured through questionnaires. Another example is the average number of complaints per case or the number of product defects.

Whereas verification focuses on the logical correctness of the modeled process, performance analysis aims at improving processes with respect to time, cost, and/or quality. Within the context of operations management, many analysis techniques have been developed. Some of these techniques "optimize" the model given a particular performance indicator. For example, integer programming or Markov decision problems can be used to find optimal policies. Analytical models typically require many assumptions and can only be used to answer particular questions. Therefore, one often needs to resort to *simulation*. Most BPM tools provide simulation capabilities.

Although many organizations have tried to use simulation to analyze their business processes at some stage, *few are using simulation in a structured and effective manner*. This may be caused by a lack of training and limitations of existing tools. However, there are also several additional and more fundamental problems. First of all, simulation models tend to *oversimplify* things. In particular, the behavior of resources is often modeled in a rather naive manner. People do not work at constant speeds and need to distribute their attention over multiple processes. This can have dramatic effects on the performance of a process [2], and therefore such aspects should not be "abstracted away." Second, various *artifacts available are not used as input for simulation*. Modern organizations store events in logs, and some may have accurate process models stored in their BPM/WFM systems. Also note that in many organizations, the state of the information system accurately reflects the state of the business processes supported by these systems because of the tight coupling between both. Today, such information (i.e., event logs and status data) is rarely used for simulation or a lot of manual work is needed to feed this information into the model. Fortunately, process mining can assist in extracting such information and use this to realize performance improvements. Third, the focus of simulation is mainly on "design" while managers would also like to use simulation for *"operational decision making"* (solving the concrete problem at hand rather than some abstract future problem). Fortunately, *short-term simulation* [110] can provide answers for questions related to "here and now." The key idea is to start all simulation runs from the current state and focus on the analysis of the transient behavior. This way a "fast-forward button" into the future is provided.

2.5.3 Process mining

Process mining is a relative young research discipline that sits between machine learning and data mining, on the one hand, and process modeling and analysis, on the other hand [3]. The idea of process mining is to discover, monitor, and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's systems.

Figure 2.21 shows that process mining establishes links between the actual processes and their data, on the one hand, and process models, on the other. Today's information systems log enormous amounts of events. Classical WFM systems (e.g., Staffware and COSA), BPM systems (e.g., BPMone by Pallas Athena, SmartBPM by Pegasystems, FileNet, Global 360, YAWL, and Teamwork by Lombardi Software), ERP systems (e.g., SAP R/3, Oracle E-Business Suite, and Microsoft Dynamics NAV), PDM systems (e.g., Windchill), CRM systems (e.g., Microsoft Dynamics CRM and SalesForce), middleware (e.g., IBM's WebSphere and Cordys Business Operations Platform), hospital information systems (e.g., Chipsoft and Siemens Soarian), and so on provide detailed information about the activities that have been executed. Figure 2.21 refers to such data as *event logs*. Most of the process-aware information systems just mentioned directly provide such

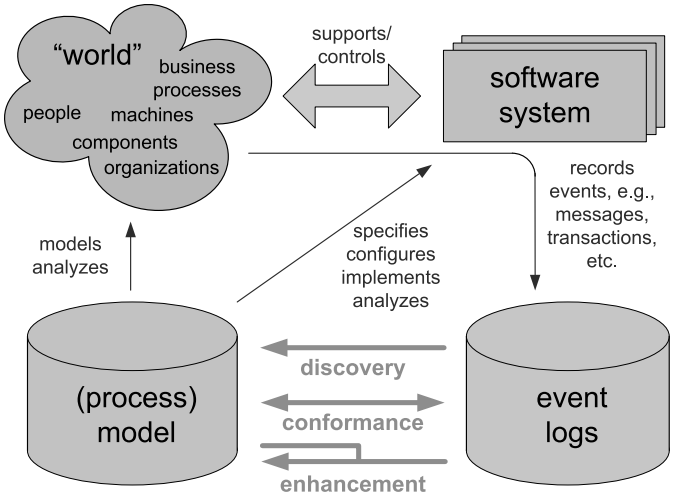


Figure 2.21
Positioning of the three main types of process mining: *discovery*, *conformance* and *enhancement*.

event logs. However, most information systems store information in less structured form (e.g., event data is scattered over many tables or needs to be tapped off from subsystems exchanging messages). In such cases, the event data exists, but some effort is needed to extract it. Data extraction is an integral part of any process mining effort.

Event logs can be used to conduct three types of process mining as shown in figure 2.21. The first type of process mining is *discovery*. A discovery technique takes an event log and produces a model without using any a priori information. An example is the α-algorithm [20]. This algorithm takes an event log and produces a Petri net. For example, given sufficient example executions of the process shown in figure 2.9, the α-algorithm is able to automatically construct the Petri net without using any additional knowledge. If the event log contains information about resources, then one can also discover resource-related models (e.g., a social network showing how people work together in an organization).

The second type of process mining is *conformance*. Here an existing model is compared with an event log of the same process. Conformance checking can be used to check whether reality, as recorded in the log, conforms to the model and vice versa. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Analysis of the event log will show whether this rule is followed. Another example is the checking of the so-called "four-eyes" principle, stating that particular activities should not be executed by one and the same person. By scanning the event log using a model specifying these requirements, one can discover potential cases of fraud. Hence, conformance checking may be used to detect, locate, and explain deviations and to

measure the severity of these deviations. An example is the conformance checking algorithm described in [110]. Given the model shown in figure 2.9 and a corresponding event log, this algorithm can quantify and diagnose deviations.

The third type of process mining is *enhancement*. Here the idea is to extend or improve an existing process model using information about the actual process recorded in some event log. While conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a priori model. One type of enhancement is *repair* (i.e., modifying the model to better reflect reality). For example, if two activities are modeled sequentially but in reality can happen in any order, then the model may be corrected to reflect this. Another type of enhancement is *extension* (i.e., adding a new perspective to the process model by cross-correlating it with the log). An example is the extension of a process model with performance data. For example, by using timestamps in the event log, one can report on bottlenecks, service levels, throughput times, and frequencies. Similarly, process models can be extended with information about resources, decision rules, quality metrics, and so on based on event data in the system's logs.

2.6 Further reading

Suggested further readings relevant to the material presented in this chapter include:

- Standard readings on Petri net theory are [40] and [101]. The issue of structured process modeling is considered at length in [74] and [72].
- [10] presents a comprehensive overview of the use of Petri nets and workflow nets for business process modeling. The application of Petri nets to WFM was first discussed in [1]. In [11], various notions of soundness are defined and corresponding analysis techniques are discussed.
- The current version of the BPMN standard is available in [97]. A high-level introduction to the use of BPMN for business process modeling is presented in [138]. A more comprehensive recent work is [123].
- The UML standard is contained in [96], and within this document, there is a comprehensive overview of activity diagrams. [51] and [80] provide an introduction to the use of various UML techniques, including activity diagrams for business modeling purposes.
- The original EPC proposal [71] is only available in German; however, the two books [119, 120] give a good introduction to EPCs and the ARIS modeling framework more generally.
- A general introduction to business process modeling is available in [63] and [136].
- For more information about the analysis of business process models, see [69]
- For more information about modeling business processes using colored Petri nets, see [70] and [19].

3 Business Process Management Systems

In this chapter, we examine the use of Business Process Management Systems (BPMSs) as a means of operationalizing business processes. In recent years, BPMSs have emerged as a generic means of supporting business processes and allow for processes in a wide variety of application areas to be automated using standard "off-the-shelf" technology. The rise of BPMS technology is consistent with trends in other areas of software engineering where generic application functionality is developed on a specialist basis and made available for reuse in the context of large software development initiatives. As a consequence, application developers are increasingly able to access high-quality software components for particular areas of functionality, thus alleviating the need to develop these components.

Figure 3.1 illustrates the evolution of supporting technology for business processes over the past five decades. In the 1960s, almost all applications were developed completely from scratch, and the available technology of the day generally resulted in the creation of monolithic applications, each fulfilling a specific purpose.

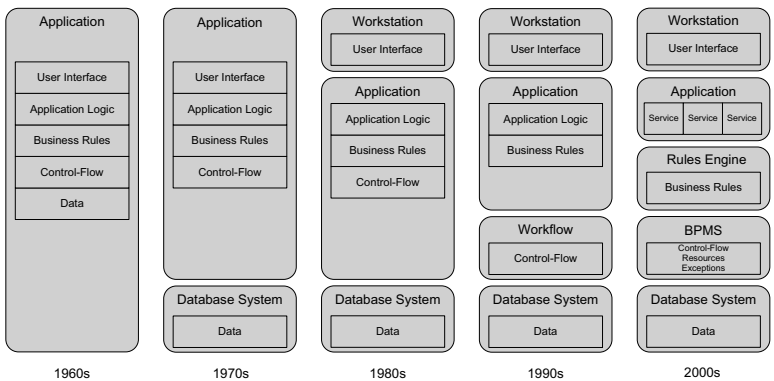


Figure 3.1
Evolution of BPM technology (from [64]).

In the 1970s, database technology became available as a generic piece of application technology, and in the 1980s, PCs, GUIs, and networking technology meant that richer user interfaces were able to be provided for software applications. The core of application functionality, however, including the majority of the associated business logic, was still coalesced in a single software module. This situation began to change in the 1990s when generic workflow technology started to become available. It provided a means of coordinating the functionality contained in a given application on the basis of a process model that was independent of the actual application. For the first time, it was possible to decouple the control-flow aspects of an application from the actual business functionality. The

development of BPM technology continued to evolve in the 2000s, with rules engine technology becoming available to handle the business rules associated with an application on an independent basis. Perhaps the most significant of recent developments in the BPM area (and the IT field more generally) is the establishment of the service-orientation paradigm as the dominant means of application construction. Service-oriented architectures (SOAs) are now the predominant means of software design and construction.

For BPM offerings, SOAs are a fundamental means of coupling the various parts of business logic within an application and also for integrating the various functional areas (UI, data, etc.) that it comprises. Moreover, the various components of a BPM application are now engineered in a more "open" way, with an increased expectation that they will need to co-exist with a wide range of third-party offerings and evolving technologies.

As workflow technology has continued to mature, it is now increasingly able to offer a broader range of facilities with which to design, deploy, and analyze business processes. In the following sections of this chapter, we will examine BPM technology in more detail, looking at how it has evolved, what form it takes, and how it aims to provide the flexibility and agility required by today's business processes. We examine three BPMS technologies — XPDL, BPEL, and case handling — as an indication of the range of BPMS solutions that are currently available, and finally we close with a brief overview of the current challenges faced in the BPM domain. First, we will look at the issues associated with business process automation.

3.1 Introduction to business process automation

In common with many other software artifacts, business processes have a distinct lifecycle associated with them. This lifecycle is shown in figure 3.2 and consists of four distinct phases. The overall lifecycle for a business process is cyclic in form, and in much the same way as the spiral model applies to the software development process, it provides the basis for the continuous improvement of a business process during its operational lifetime.

The first stage in the business process lifecycle is *process design*, during which a candidate business process is first designed or redesigned based on previous incarnations. This typically involves capturing the operational details of the process using some form of modeling notation. Ideally sufficient details of the process are recorded to facilitate its direct enactment; however, this relies on the use of an executable modeling formalism, such as that provided by a workflow offering. Generally, the design is captured using notations such as BPMN, EPCs, or UML activity diagrams, and it takes the form of a conceptual process model. Further detail (which is often added subsequently) is generally required to enable it to be executed.

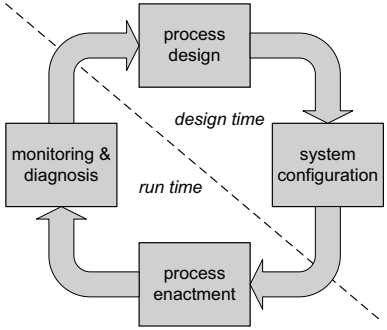


Figure 3.2
Business process lifecycle.

The second stage in automating a business process is *system configuration*. In contrast to traditional software development activities, the deployment of a business process is generally based on the configuration of BPMS software and usually only involves minimal amounts of direct programming effort. This is a consequence of some assumptions about the approach to process deployment that are embodied in the BPMS software. Typically the configuration process is done using facilities provided by the BPMS tool and involves activities such as importing or creating the candidate process model using some form of graphical editor, defining data objects and their interrelationship with the control-flow aspects of the process model, and defining resources and the distribution of work within the process to those resources. Figure 3.3 outlines the standard architecture for a BPMS. It consists of two main parts:

- a *BPMS engine*, which handles the execution of a business process on a centralized basis and manages current process state, working data, work distribution, and so on for each process instance that is initiated; and
- a *worklist handler* for each user who is involved in a process, which advises them of the work that needs to be undertaken and provides them with the opportunity to interact with the BPMS engine to advise of their work preferences and manage the scheduling and completion of work to which they have committed.

Process design and system configuration are *design-time* activities that involve the development of a process model that has sufficient detail to allow it to be directly executed. In contrast, the next two stages in the lifecycle are *runtime* activities, which occur once the model of the business process that has been developed is actually under execution. Traditionally, in many systems, there was a distinction between the design-time and runtime process models, and they were handled by distinct components of the BPMS software. The design-time process model was used to facilitate the overall definition and capture of the

business process and was often undertaken using a graphical design tool or editor. Once finalized, the model was converted to a runtime process model that was used as the basis of process execution by the BPMS. More recently, with the increased use of executable process models such as BPMN, this is becoming less of an issue, and design time and runtime models are effectively one and the same.

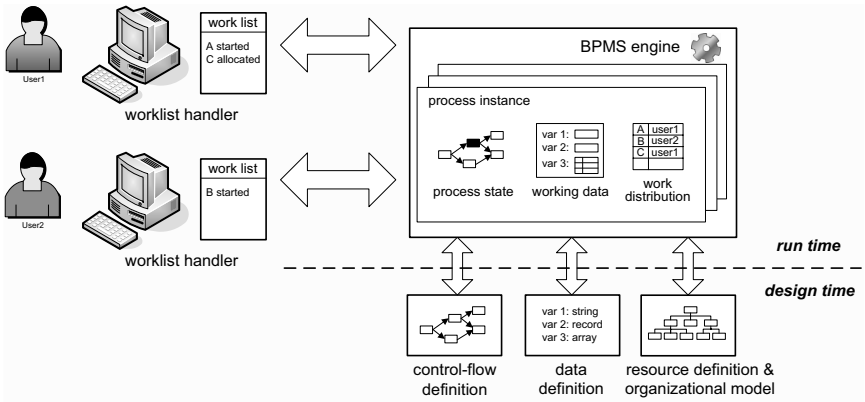


Figure 3.3
Overview of an operational BPMS.

The next stage in the business process lifecycle is *process enactment.* This involves the actual execution of the candidate process model and the routing of associated work items to users. This is done in accordance with a defined control-flow model in conjunction with data and resource definitions and an associated organizational model, although the specific details of how this is done vary on a system-by-system basis.

The final phase in the business process lifecycle is *monitoring & diagnosis*, in which the operation of the process is tracked and the resultant information is used as the basis for further refinements to the business process model. These may be dynamic changes intended to improve performance or may be more pervasive in nature and focus on adapting the process to changes that have been identified in the operational environment, providing support for new (sub-)processes or leveraging new technologies that have become available.

In this section, we have overviewed the main characteristics of the business process lifecycle and the way in which they are supported in a BPMS. The structure of WFM systems and BPMSs has been discussed in detail, and in the next section, we look at these architectural considerations in more depth.

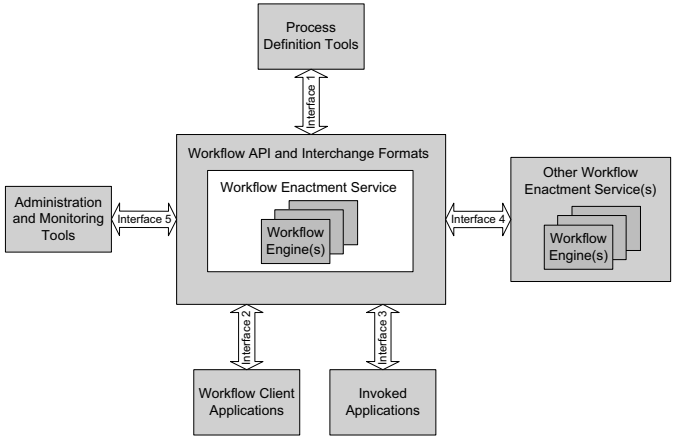


Figure 3.4
Workflow reference model (from [143]).

3.2 Architectural considerations

In an attempt to provide some direction and standardization to the workflow area, the Workflow Management Coalition (WfMC), a loose consortium of vendors, research organizations, and workflow users, was formed in 1993. In 1995, it introduced the Workflow Reference Model illustrated in figure 3.4 in an effort to align terminology in the area and define a series of interfaces for various aspects of workflow systems that vendors could adopt, thus promoting the opportunity for interoperability between distinct offerings.

The workflow reference model defined five key interfaces for workflow systems:

- *Interface 1* (Process definition tools), which details a generic process definition language (Workflow Process Definition Language (WPDL)) that describes common workflow elements and their interrelationships;
- *Interface 2* (Workflow client applications), which describes the interactions between a workflow engine and a client application (e.g., a worklist handler);
- *Interface 3* (Invoked applications), which defines an interface for invoking remote applications;
- *Interface 4* (Other workflow enactment services), which provides an interface for distinct workflow systems to interact with each other; and
- *Interface 5* (Administration and monitoring tools), which focuses on the specification of the format and interpretation of the audit trail details associated with workflow execution.

Although it met with varying degrees of criticism and vendor support among the workflow community, the workflow reference model had a significant impact on the overall technological foundations of workflow, and for many years it remained the only "standard" in the area with any degree of broad acceptance. More significantly, it reinforced the notion of what constituted the architecture for a workflow system. The workflow systems that came to market in the 1990s for the most part operated as standalone applications, with the workflow application providing all of the process enablement and user interface functionality other than data persistence, which was typically provided by distinct database system software. In recent years, however, there has been a trend away from "closed" workflow applications to a new generation of more "open" BPMSs, which incorporate workflow technology and position themselves more as solutions for embedding process support in an organizational environment. Figure 3.5 illustrates the broad range of potential integration options offered by this technology. In essence, it focuses on providing coordination of process state and allows all other aspects of business process operation to be delivered by other software applications.

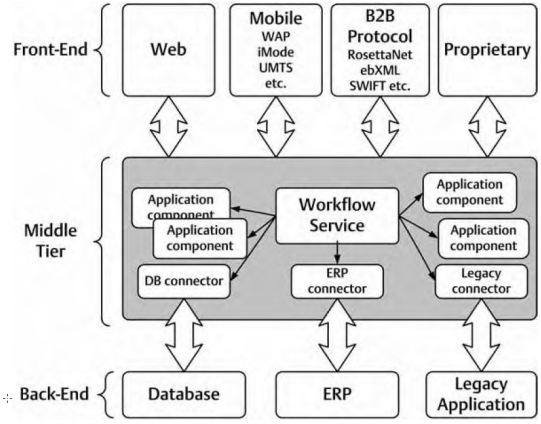


Figure 3.5
Embedded workflow as an integration technology (from [88]).

3.3 A brief history of BPMS technology

In this section, we provide a brief overview of the history of BPMS technology. This starts with early work in the area of office information systems and, through workflow systems, takes us to contemporary BPM systems. Figure 3.6 illustrates this timeline, showing the

main technology streams and some prominent examples of actual systems in each of these areas.

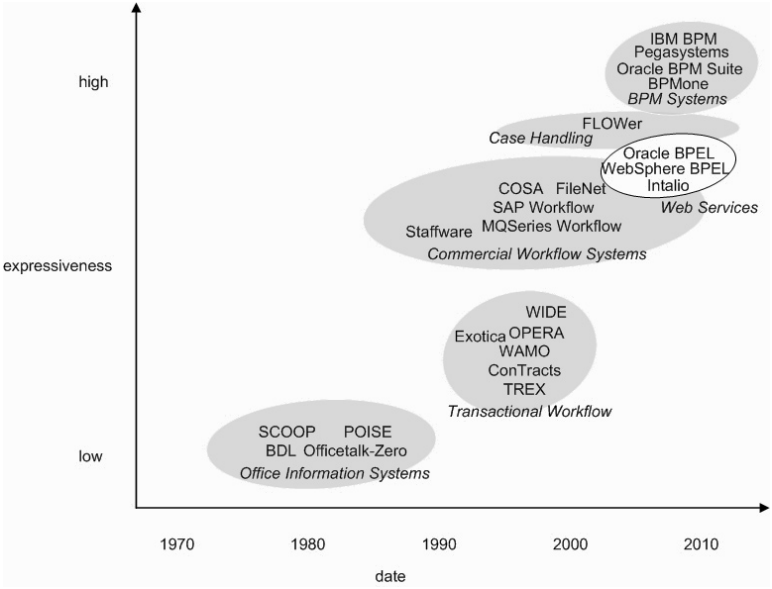


Figure 3.6
BPM technology timeline.

3.3.1 Office information systems

The antecedents of the BPMS research area lie in the field of office information systems. The promise of automating aspects of the office function (such as document and work distribution, communication and retention of work-related information, etc.) triggered several independent research initiatives into the formal definition of office procedures. Zisman [151], Holt [66], and Ellis [62] all proposed state-based models of office information systems based on Petri nets. Gibbs and Tsichritzis [56] documented a data model for capturing the "structure and semantics of office objects." There were also several prototype systems developed, including SCOOP [151], POISE [36], BDL [62], and Officetalk-Zero [50], although none of these progressed into more widespread usage. A variety of reasons are cited for their ultimate failure [42, 49], including the inherent rigidity of the resultant systems interfering with rather than expediting work processes, the inability of the process modeling formalisms to deal with changes in the application domain, the lack of comprehensive modeling techniques, and more generally the relative immaturity of the hardware and networking infrastructure available to support them.

3.3.2 Transactional workflow

One of the first attempts to establish generalizable workflow facilities was based on the notion of "transactional workflow," which sought to extend database technology to support the notion of long-duration transactions. This approach to workflow was based on the idea that real-life processes were composed of activities that could best be enacted using related sequences of transactions. Although few of these offerings in this area progressed beyond the research stage, they had a significant impact on the architecture of later workflow systems, particularly in the area of exception handling. Notable contributions included WAMO [47], which provided the ability to specify transactional properties for tasks that identified how failures should be dealt with; ConTracts [105], which proposed a coordinated, nested transaction model for workflow execution allowing for forward, backward, and partial recovery in the event of failure; and Exotica [26], which provided a mechanism for incorporating Sagas and Flexible transactions in the commercial FlowMark workflow product. TREX [124] proposed a transaction model that involves treating all types of workflow failures as exceptions. OPERA [60] was one of the first initiatives to extend the transaction paradigm to incorporate language primitives for exception handling in a workflow system, and it also allowed exception handling strategies to be modeled in the same notation as that used for representing workflow processes. WIDE [33] incorporated transactional primitives in the workflow language and developed a comprehensive language — Chimera-Exc — for specifying exception handling strategies in the form of Event-Condition-Action (ECA) rules. METEOR [122] is a CORBA-based WFM system supporting transactions. MOBILE [68] is another example of a prototype WFM system that influenced later commercial WFM systems.

3.3.3 Commercial workflow systems

Commercial workflow technology evolved without such a tight emphasis on transactional aspects and consequently met with greater success. It soon achieved the aim of providing a general purpose enactment vehicle for processes. Initially focused on specific domains such as document routing, image management (especially in the medical diagnostics area), and advanced email applications for supporting group work, these offerings became increasingly configurable and domain independent and led to an explosion of offerings in the commercial and research spheres. Indeed by 1997, it was estimated that there were more than 200 commercial workflow systems in existence [8]. As a software domain, it was an extremely volatile area, and many offerings failed to achieve the critical mass required for long-term survival and either disappeared from the market or were merged into competitive products. Increasingly, it became clear that those offerings that were successful in achieving long-term viability were those that specialized in a particular area or demonstrated a unique capability. Staffware achieved success through its ability to facilitate rapid deployment of workflow solutions, SAP Workflow was an early example of a workflow offering

that effectively coordinated the operations of other complex software offerings (in this case, the SAP R/3 offering) and integrated closely with other third-party offerings, COSA demonstrated the viability of a deterministic workflow offering based on formal foundations (in this case, Petri nets), and MQSeries Workflow/WebSphere provided an industrial strength workflow solution capable of scaling to meet the needs of enterprise users.

The range of offerings that can be classified as workflow systems is extremely broad. Georgakopoulos et al. [55] offered a characterization of the area as a "workflow umbrella," in which workflow technology can be viewed as a continuum ranging from human-oriented workflow, which supports humans coordinating and collaborating on tasks that they are ultimately responsible for undertaking, through to system workflow, which undertakes computationally intensive tasks on a largely automated basis. Along this spectrum are a variety of forms of workflow technology, including Computer Supported Cooperative Work (CSCW) or groupware, commercial WFM systems, and commercial Transaction Processing (TP) systems. Another perspective is offered in figure 3.7, which classifies process-enabling technologies on the basis of whether they tightly adhere to a process model (tightly framed) or not (unframed) and what the automation intent of the system is (e.g., P2P: person-to-person, P2A: person-to-application, or A2A: application-to-application). This delineates a range of distinct technology groupings, including various types of workflow technology.

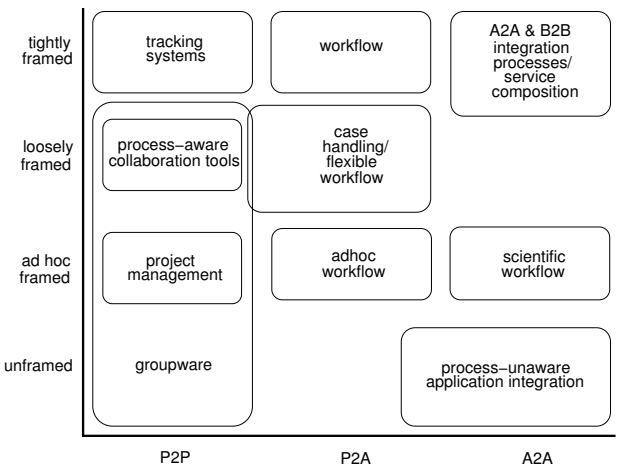


Figure 3.7

Classes of business process technologies (from [42]).

3.3.4 Case handling

Around the same time as the explosion of commercial workflow technology, the case handling paradigm was also established [21]. This specific approach to business process enactment is based on the idea that processes are driven by data-flow rather than control-flow. This approach to process enablement is particularly suited to processes that are highly data intensive, execute over a long duration, and are largely under the auspices of a single individual or group, such as the evaluation of an insurance claim or the registration of a newborn child. The leading offering in this area is the BPMone product from Perceptive Software. The specifics of this offering are discussed in more detail in section 3.4.3. Case handling is one of several approaches to make WFM/BPM technology more flexible. Examples of WFM systems providing innovative flexibility support are ADEPT and Declare (see section 3.5).

3.3.5 Web services

Web services composition languages have been an area of significant research over the past few years as more flexible and lightweight means were sought for providing business process support, particularly where these processes involve distributed participants or services provided by distinct vendors. Increasingly, the focus of these languages has moved beyond the traditional focus of the workflow system to the broader class of execution environments referred to as Business Process Management Systems (BPMSs).

One of the foci of standards initiatives in the business process area has been to define a modeling language for business processes that contains sufficient detail for it to ultimately be enacted. In 2002, the Business Process Management Institute (BPMI) released the Business Process Modeling Language (BPML), a standard for describing business processes and their constituent activities at varying levels of abstraction. Accompanying this proposal in draft form was the Business Process Modeling Notation (BPMN), a graphical notation for expressing business processes (see chapter 2 for more details on BPMN).

Although BPMI ultimately withdrew support for BPML, the graphical modeling notation BPMN received widespread attention, and despite its initial focus on the control-flow perspective of business processes, its influence has extended to the WfMC standards, and Interface 1 is now implemented as XPDL 2.2, which is essentially an XML-based serialization and interchange format for BPMN 2.0.2.

The most pervasive initiative in this area has been BPEL, a workflow-based web service composition language that has a broader range of capabilities in comparison to other initiatives in this area. A particular feature of BPEL is that it is designed to allow for the specification of both abstract and executable processes. Although mainly focused on the control-flow perspective, it also incorporates concepts from other perspectives such as data-passing, event and exception handling, messaging, and constraint enforcement.

Two of the most significant offerings in this area are Oracle BPEL Process Manager and WebSphere Process Server. Oracle BPEL Process Manager and WebSphere Process Server are enterprise-grade offerings that form part of the Oracle SOA Suite and WebSphere Process Server suites, respectively. Both offerings are designed to co-exist with the wide range of other development tools (database, application, webserver technology, etc.) that fall under these umbrellas, thus enabling workflow technology to be integrated with other corporate applications. In an attempt to better support human interactions within the BPEL framework, the BPEL4People extension has been proposed and is supported in commercial offerings such as Oracle SOA Suite and BPM Suite.

3.3.6 Business process management systems

As business process technology continues to mature, there is an increasing trend for offerings to provide support for all aspects of the BPM lifecycle. Whereas early workflow offerings simply provided a process design tool (generally with some form of graphical interface) and a process execution environment, recent BPMS offerings are also including support for various forms of process analytics (including process monitoring and business activity monitoring), content management, business rule support, and collaboration tools. In effect, a BPMS offering provides total support for the design, deployment, monitoring, and ongoing enhancement of business processes.

Pegasystems SmartBPM Suite, IBM Business Process Manager, Oracle SOA Suite and Oracle BPM Suite are leading offerings in this area. Oracle is particularly comprehensive in its process modeling and enactment capabilities supporting process specification in both BPEL and BPMN in an integrated application infrastructure together with integrated support for human processes as described in the BPEL4People and WS-HumanTask proposals, and business rules and business activity monitoring (BAM) functionality

BPMone is capable of supporting both workflow-style processes as well as its unique case handling approach. It incorporates a range of process management facilities relevant to the lifecycle of business processes, including simulation, process mining, and visual analytics.

3.4 Overview of popular techniques

Multitudes of business process definition languages have been proposed over recent years. In this section, we look at three significant initiatives that focus on the definition of business processes in sufficient detail that they can be directly executed. XPDL and BPEL are language initiatives operating under the auspices of the WfMC and OASIS industry

bodies, respectively. Case handling is an alternative approach to business process enablement, which performs well in data-intensive business processes. For each of these three techniques, we look at a candidate offering and examine the capabilities it provides for business process enablement.

3.4.1 XPDL

XML Process Definition Language (XPDL) is a process definition language developed by the WfMC. It grew out of the Workflow Process Definition Language (WPDL) effort, which aimed to provide a standard for the interchange of process definitions, thus fulfilling the requirements for Interface 1 of the Workflow Reference Model. The first release of XPDL was finalized in 2002 and received significant industry focus; however, although many vendors claimed to provide support for the language, few actually made any serious efforts to achieve compliance. As a consequence, the language remained largely dormant and had little impact on industry trends.

This situation changed in 2004 when the WfMC formally adopted BPMN (up until that point perceived as a competing initiative for representing business processes) as the graphical formalism it would use for standardizing the visualization of business processes. As part of this effort, it made the decision to extend XPDL to allow it to represent all of the concepts contained in BPMN, including not only the semantic content of the BPMN model but also any relevant graphical layout information. The latest revision — XPDL 2.2 — was finalized in 2012 and caters to the wide range of concepts embodied in the BPMN 2.0.2 modeling standard.

XPDL uses an XML-based syntax based on XML Schema, and consequently process definitions are captured in a textual format. The emphasis of the current version is to provide a serialization of the BPMN modeling language that enables BPMN models to be exchanged between distinct BPM modeling tools, a concept that falls outside the purvue of the BPMN specification. One of the issues that comes to the fore when attempting to support the portability of process definitions is dealing with the additional executable details that specific vendors allow to be added to a BPMN model in order for it to be able to be directly executed by their toolset. Obviously, these extensions are vendor specific and do not form part of the "abstract" BPMN model that is able to be exchanged, and for this reason, in the most recent XPDL revision, considerable effort has been put into delineating the range of abstract concepts that are intended to be portable from the executable details that are not. XPDL 2.2 includes the notion of *Portability Conformance Classes* describing the various abstract concepts that are able to be shared and defines three distinct portability levels: *simple*, *standard*, and *complete*, which enable a specific modeling tool to assert the extent of model interoperability that it is able to support.

A significant debate in recent years has centered on the ability of a BPMN model to be executed. Two distinct viewpoints have formed on the matter: those who believe that a

BPMN model is intended as a purely abstract modeling notion and those who believe that the modeling formalism can be made more precise and enhanced to the point where it is able to be directly executed. Despite this debate, the use of executable BPMN is now becoming widely adopted as the standard business process modeling and execution language in commercial offerings. Because the focus of XPDL is now essentially to provide a serialization of BPMN, it does little more than reflect the concepts agreed on and embodied in BPMN, and hence XPDL is beginning to wane in attractiveness as an execution language. In the following section, we introduce the first significant attempt at standardizing a BPM execution language: BPEL.

3.4.2 BPEL

Web Services Business Process Execution Language (BPEL) is an executable language for describing web service interactions. It has its genesis in the WSFL and XLANG initiatives championed by Microsoft and IBM, respectively, who recognized the need for such a language and, faced with the increasing prominence of another initiative in the area (BPML), decided to amalgamate their efforts. The result was BPEL4WS, which was first released in 2003 and has met with broad support from vendors. Soon after, in conjunction with a group of vendors, the BPEL proposal was submitted to OASIS as part of a formal standardization effort, and they have overseen its development from this point. The current release is termed WS-BPEL 2.0, in line with other web services standards that OASIS administers.

BPEL is intended as an orchestration language, and consequently it describes web service interactions from the viewpoint of a single business process. It can be used at two distinct levels, allowing for the specification of either *abstract* or *executable* business processes depending on the level of detail that is captured about a candidate process.

Unlike earlier attempts at defining web services, which were limited to stateless interactions, BPEL is deliberately intended as a means of describing stateful, conversation-oriented business processes that are based on web services. A BPEL process is described in terms of XML. The high-level structure of a BPEL process is illustrated in Listing 3.1. It contains a number of language elements prior to the definition of the activity section, which describe the actual mechanics of the business process.

The purpose of each of these elements is as follows:

- `<extensions>` describe language extensions used in the context of the BPEL process. These may range from new types or elements through to new activity definitions and provide a mechanism for restricting or further extending current runtime behavior. Each `<extension>` element identifies the namespace in which the extension is located.
- `<import>` provides a means for a BPEL process to include any externally defined XML Schema or WSDL definitions on which it relies. These may include partner link types (the notion of partner links is described below), variable properties, and property aliases described elsewhere.

```
<process>
  <extensions> </extensions>
  <import> </import>
  <partnerLinks> </partnerLinks>
  <messageExchanges> </messageExchanges>
  <variables> </variables>
  <correlationSets> </correlationSets>
  <faultHandlers>? </faultHandlers>
  <eventHandlers> </eventHandlers>

  activity

</process>
```

Listing 3.1: BPEL process definition structure.

- <partnerLinks> are a key feature of a BPEL process. They describe the individual conversational links that a process has with various external partner services during its operation. The services fulfill specific functional requirements that are required during process execution. Each link identifies the service interaction(s) that occur between the process and its partner and the role that each of them is expected to perform during the course of an interaction.
- <messageExchanges> are used to distinguish between distinct conversational interactions that an instance of a BPEL process may have with a partner service. Because all conversations are based on incoming message receival activities and corresponding reply activities within a BPEL process, and it is possible for distinct conversations to occur with the same partner simultaneously, individual <messageExchange> constructs facilitate the pairing of receival and reply activities, thus enabling distinct conversations to be disambiguated.
- <variables> provide the containers for storing the state information relevant to a process. This can be messages received from or intended for transmittal to a partner or it can be data elements used during the execution of a business process.
- <correlationSets> provide a means of tying related incoming and outgoing messages from a business process to a given partner service into a conversation. A <correlationSet> consists of a list of message properties that remain fixed during the course of a given conversation and can therefore be used for correlation purposes. When the first message in a conversation is sent, the values of the message properties are fixed. By including these properties in subsequent message interactions, the correlation of messages into a conversation by both the BPEL process and the partner service is possible.
- <faultHandlers> provides a means of associating error handling strategies with a BPEL process. Termed fault handlers, these strategies may apply to a specific scope within a process or to the entire process. They can respond to nominated faults generated

by the execution of the BPEL engine or raised during the execution of activities within the process. When a fault is detected during the execution of a process, execution is suspended within the scope of the process to which the fault handler corresponds and control passes to the relevant fault handler. Once invoked, fault handlers have a range of options at their disposal for dealing with a detected fault: they can handle it themselves and then resume execution, they can invoke a compensation handler to try and mitigate the effects of the fault, they can trigger a fault at a higher level in the process, or they can cause execution of the associated scope or even the entire process to be halted.

- `<eventHandlers>` are analogous to fault handlers; however, they have a distinct purpose and consequently operate in a different manner. Event handlers provide a means of responding to events arising during the operation of a BPEL process that need to be dealt with but are not necessarily error-related (e.g., a timeout occurring). Event handlers can be associated with either a specific scope within a process or the entire process. They can be triggered by either the receipt of a specific message type or the occurrence of a nominated event, such as an alarm. When invoked, event handlers run concurrently with the process. They can deal with the detected event in a number of ways, including resolving the issue themselves, canceling execution in the associated scope, propagating the detected event to a higher level in the process, or terminating the process.

As shown in Listing 3.1, the final part of a BPEL process definition is the activity. At the top level, a BPEL process consists of a single activity; however, this can contain an arbitrary number of nested subactivities. Activities can be either *basic* or *structured*. Basic activities correspond to the fundamental actions undertaken during process execution. The following basic activities are supported by BPEL:

- `<invoke>` provides a means of invoking a partner service.
- `<receive>` allows an inbound message to be received for processing.
- `<reply>` enables a response to an inbound message to be sent to a partner service.
- `<assign>` enables data to be copied from one variable to another.
- `<wait>` causes execution to be delayed until a nominated timeframe has expired.
- `<empty>` corresponds to the “do nothing” activity.
- `<extensionActivity>` allows a task implementation to be extended by some form of new technology.
- `<exit>` causes the process to terminate immediately.
- `<throw>` generates a fault during process execution.
- `<rethrow>` enables a fault to be redirected to another fault handler.
- `<compensate>` initiates compensation handling for all inner scopes.
- `<compensateScope>` initiates compensation handling for a nominated (inner) scope.
- `<validate>` supports the validation of the values of variables against their data types.

Structured activities capture control-flow logic and may also include other basic and/or structured activities. The following structured activities are supported:

- `<sequence>` provides a means of indicating that a group of enclosed activities should be executed sequentially.
- `<if>`, `<elseif>`, `<else>` provides a mechanism for denoting that activities should be executed on a conditional basis.
- `<while>` provides a means of executing an enclosed activity on a conditional (pretested) basis zero or more times.
- `<repeatUntil>` provides a means of executing an enclosed activity on a conditional (posttested) basis one or more times.
- `<forEach>` provides a means of executing an enclosed activity a specified number of times either sequentially or concurrently.
- `<pick>` allows one out of several specified activities to be executed depending on which of a group of associated events occurs first.
- `<flow>` enables one or more specified activities to be executed concurrently.
- `<scope>` provides a means of defining a nested activity that is able to be compensated independently of other activities.

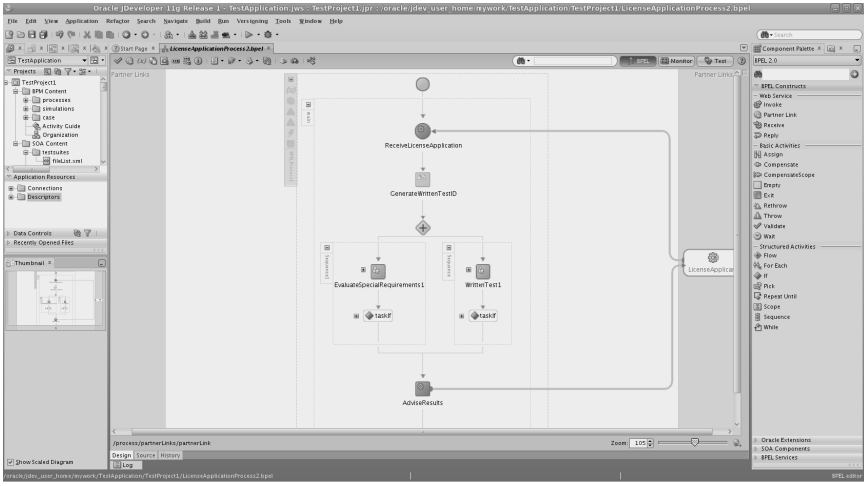


Figure 3.8
jDeveloper: the graphical editor used with Oracle BPEL.

A number of BPEL engines are available in the marketplace, ranging from open-source initiatives through to commercial engines intended for production usage. In this book, we will examine Oracle BPEL Process Manager, a widely used offering that is effectively integrated with the broader Oracle SOA Suite (a range of software incorporating products

in the database, middleware, and BPM areas). An innovative feature of the Oracle SOA Suite offering is the jDeveloper graphical editor; it provides for the specification of business processes. Figure 3.9 shows the main screen for the editor, which provides an interactive graphical means of specifying BPEL processes and all of their associated elements and linkages with other partner services. It significantly reduces the effort associated with the specification of a BPEL process by providing a series of palettes of process constructs that can be used in the design of BPEL processes as shown in figure 3.9.

Figure 3.9
Process construct palette in jDeveloper.

Four main palettes are available for use in developing BPEL processes:

- *BPEL Constructs*, which correspond to language constructs in the current BPEL version.
- *Oracle Extensions*, which provide a series of additional (Oracle specific) constructs that can be incorporated directly in BPEL processes to enable features such as specification of assertions, external notification of events via various communication technologies (email, fax, IM, voice, etc.), supporting persistent storage of the state of long-duration processes, embedding Java code in a process, re-executing activities, and supporting coordination between master and detail processes.

- *SOA Components*, which provide four additional (Oracle-specific) service components enabling BPEL processes to interact with other distinct BPEL processes, utilize business rule functionality, support human interaction with processes, and incorporate mediation facilities for managing data interactions with other components in a composite application.
- *BPEL Services*, which provide facilities for BPEL processes to communicate and interact with other web-based services and applications via a range of distinct adapters and binding mechanisms.

From our perspective, the most interesting of the extensions to Oracle BPEL is the *Human Task*, which enables activities with workflow-like behavior to be embedded within a BPEL process. These activities can be routed to a specific user or group of users for subsequent execution. By extending the fundamental BPEL language in this way, its usefulness is significantly enhanced, and the criticism that it faced in regard to its lack of support for human processes is suitably addressed. The interaction facility between the BPEL process and the person undertaking the activity is illustrated in figure 3.10. Activities are presented to individual users via a worklist application.

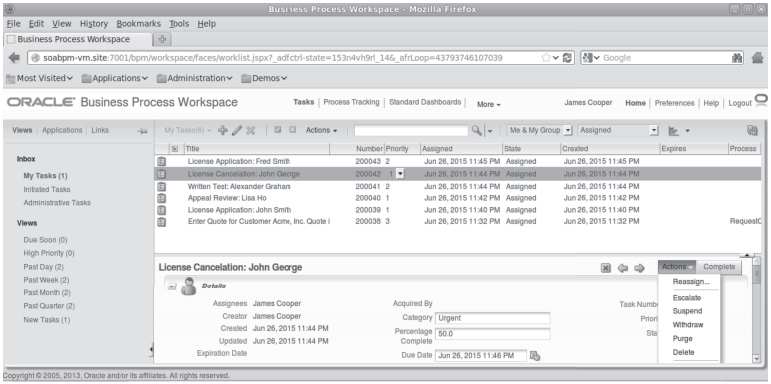


Figure 3.10
Oracle Business Process worklist.

The worklist serves as the interface between the human user and the executing process. It allows tasks to be routed to specific users and their execution to be managed through to conclusion. Once triggered, each human task in an Oracle BPEL process transitions through a sequence of states as illustrated in figure 3.11 as its execution progresses to a successful conclusion or some other final state. Tasks may be routed directly to one or more individual users, groups, or roles. The identity of these participants may be statically specified in the process model or determined dynamically based on business rules or some other form of programmatic assignment. It is even possible to utilize the services of an external routing service for this purpose. A number of distinct task assignment approaches

— known as participant types — are supported in order to cater for the range of ways in which individual tasks are assigned and undertaken by process participants:

- *Single approver* allows a task to be routed to a single user, group, or role; however, it assumes the task will be claimed and completed by a single user. Once claimed, it is withdrawn from the task list of other users to whom it may have also been routed.
- *Parallel* enables a task to be routed to and worked on by several users simultaneously. Task completion is determined on the basis of a group vote. Various voting configurations are supported.
- *Serial* allows a task to be routed sequentially to a set of users, each of whom may work on it in turn. This approach caters to approval policies such as management chain, where a number of users in an organizational hierarchy each undertake and confirm parts of a task.
- *FYI (For Your Information)* differs from the previous participant types in that it supports users receiving notifications about a task and being able to contribute to its information content (e.g., by adding comments or attachments) but does not involve the task being actually routed to the user for execution nor can the user have any impact on the overall progress of the task.

Where tasks are routed to the worklists of multiple users (e.g., by a group or role allocation), they must subsequently be claimed by individual users before they can be completed. As soon as they are assigned, tasks are assumed to be active (i.e., there is not a distinct start action). However, there is a distinct completion action, and this can be configured in a number of ways for each task depending on the specific routing approach that has been selected.

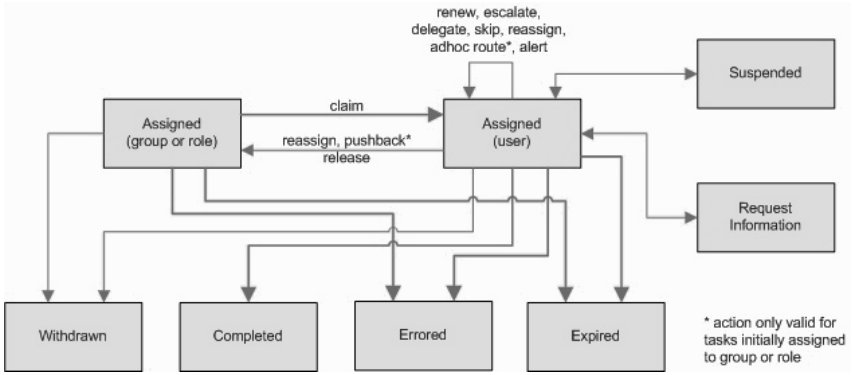


Figure 3.11
Task lifecycle in Oracle BPEL.

As can be seen, Oracle BPEL provides a broad range of facilities for configuring the routing of tasks and the range of task support facilities available to users. Furthermore, all

of these features can be programmatically extended to further refine their capabilities and applicability to the wide range of situations in which process support may be required. The following three chapters discuss the various features and capabilities of Oracle BPEL in greater detail in the context of the control-flow, data, and resource patterns.

3.4.3 Case handling

One of the frequent criticisms of classical WFM/BPM technology is its focus on routing as the main means of driving work through a process model. The execution of a given case is based on the traversal of the thread of control though a process model using a predefined set of rules and constraints. Tasks in the process are only triggered when they receive the thread of control. There is no direct consideration of other important workflow factors such as the availability of data or resources when determining the execution sequence for activities in a given case, a phenomenon known as *context tunneling*. The emphasis that traditional workflow places on breaking down processes into a set of activities that can be routed to users for execution also has other difficulties as often there is a discrepancy between the way in which work activities are described by the process and the way in which users actually perceive and execute them. Typically users undertake work at varying levels of granularity. The classical process model breaks down work into fixed chunks, whereas users tend to dynamically compose activities from lower level actions.

In a workflow system, work is typically routed to users by the system on a preemptive basis, and users are not empowered to choose what they would like to do from work that needs to be done. Moreover, the routing of work serves two distinct functions: *distribution* and *authorization*. Individual workers only see the work that they are authorized to undertake. It is not possible for them to gain a broader view of their part in the conduct of the overall process.

These limitations seriously constrain the overall flexibility of workflow technology and its ability to effectively implement a wide variety of processes. As a remedy to these issues, *case handling* has been proposed as a means of supporting knowledge-intensive business processes [21]. Case handling has a number of core features that alleviate many of the aforementioned issues. In particular, it seeks to:

- avoid context tunneling by providing workers with access to all of the information relevant to a case rather than just the data for the work item at hand;
- base the decision on the work to be executed on information that is currently available within the case rather than on which activities have completed execution;
- impose a distinction between the act of work distribution and the concept of authorization; and
- allow users to access and amend data elements before or after the tasks to which they correspond have been executed.

Case handling is based on the fundamental notion that the *case* is the most important aspect of process execution, not the individual tasks that make up a process or the way in which work associated with it is distributed. As in a traditional workflow, a process is assumed to be made up of tasks, and there are directed arcs between them indicating precedence relations. However, unlike traditional workflows, the precedence relationship does not dictate the order of execution or the way in which tasks will be routed; it only provides a guide as to the dependencies that exist in the ordering of tasks in the process. Later tasks typically have some sort of execution dependency on or information requirement from earlier tasks. Moreover, the existence and value of data elements influence the sequence of tasks undertaken in a specific case. In contrast to other traditional BPM systems, users can also partially execute tasks or undertake them on an incremental basis.

BPMone is an integrated business process management environment that supports the full business process lifecycle. It includes support for process design, execution, analysis, and enhancement and has a particular focus on case handling as a means of conducting individual process instances. It integrates a number of tools that previously had an independent existence and area of focus, including Protos (process design and analysis), Activate (workflow), and FLOWer (case handling). BPMone also provides simulation facilities and supports process mining both during the early design stages and while controlling processes. Moreover, cases executed by the system can be analyzed using visual analytics. A key characteristic of the BPMone product is the way in which it integrates the process, data, and resource aspects of business processes. It supports processes definitions at two levels: (1) higher level business processes that can be used for process analysis and workflow-style process execution, and (2) more detailed business processes that contain sufficient detail for their execution to be undertaken on a case handling basis and support a greater range of options for adapting the specific process actions undertaken for individual cases.

Figure 3.12 illustrates the higher level BPMone process for the license application process discussed in chapter 2. It shows the way in which the BPMone design environment integrates the various aspects of the process from the control-flow, data, and resource perspectives. In general, the semantics of the model are much the same as the Petri net process model illustrated in figure 2.9. BPMone is able to handle both a deferred choice (i.e., a choice made in the operating environment of the process) and an explicit choice (i.e., a choice made by a user) between specific paths in the process as illustrated by the branches after the *Practical test pending* and *Make recommendation* nodes, respectively.

The higher level process models in the BPMone environment can be used directly for process analysis and execution but assume a more traditional workflow-style approach to process execution, where a case proceeds basically in terms of the task sequence in the associated process model. The more detailed process models supported by BPMone (known as *case type* models) support the full range of case handling options when dealing with the tasks associated with a given case.

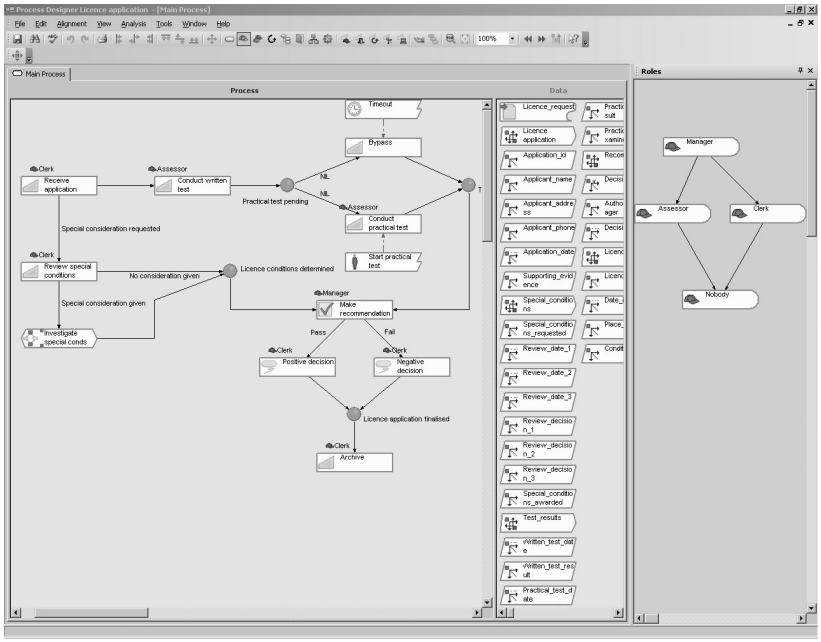


Figure 3.12
Process design using BPMone.

Case-type models in BPMone are comprised of a series of elements connected by directed arcs in a hierarchical graph structure as with other process modeling formalisms. Figure 3.13 illustrates the various constructs that make up a BPMone process model. Note that BPMone uses the term *plan* to refer to an individual process model and *step* to refer to an individual task or activity within a plan. The function of each of these constructs is as follows:

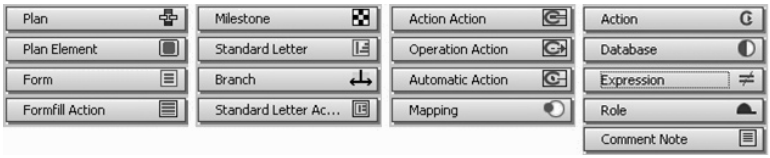


Figure 3.13
Language elements in BPMone case-type process models.

- the *Plan* construct corresponds to a subprocess embedded within a process model. A plan may have several distinct forms depending on whether it executes once, a number of times concurrently, or sequentially, or whether it contains alternate decision paths that may be selected by a user or the system at runtime;

- the *Plan Element* construct defines a step within a process;
- the *Form* construct identifies a form that is used for interacting with process participants during execution. This may be for gathering data or to display information relevant to case execution. An example of the forms supported within BPMone is shown in figure 3.15;
- the *Formfill Action* construct defines a step where part or all of a form is filled out;
- the *Milestone* construct defines a step within a plan that corresponds to a waiting moment where execution is delayed pending data availability or some other form of event;
- the *Standard Letter* construct defines a form letter that can be populated with data during execution. Letters provide a means of interacting with users in regard to more complex data conditions or where the user is not able to interact directly with the system;
- the *Branch* construct defines a decision point in a process;
- the *Standard Letter Action* construct describes a step where a standard letter is actually generated;
- the *Action Action* construct describes the action taken when a button on a form is selected;
- the *Operation Action* construct describes a step that is based on the execution of a module function within the BPMone system or an external function residing on the same server;
- the *Automatic Action* construct describes a step that is based on the automatic start and execution of a module function within the BPMone system or an external function residing on the same server;
- the *Mapping* construct identifies data retrieved from or stored in a database during execution;
- the *Action* construct defines a function in a module, an application or a rule that is utilized during execution;
- the *Database* construct defines an internal or external database in which information is stored or retrieved from during execution;
- the *Expression* construct defines a parameter or expression used during execution;
- the *Role* construct identifies a role for a user utilized within the process model. Roles form a hierarchy (as can be seen in figure 3.12) and define the basis for various privileges during execution (e.g., completing, redoing or skipping a step); and
- the *Comment Note* construct, which provides a means of annotating a process model.

Although there is generally a correspondence between the high-level BPMone model and the case-type model for a given process (and indeed there is a function within the tool to convert the former to the latter), the latter typically includes a great deal more information in regard to the way that individual process steps should be managed. They also include comprehensive support for forms and external application and database linkage. Figure 3.14 illustrates the case-type model corresponding to the license application process. A number of variations are evident when compared with the high-level model in figure 3.12.

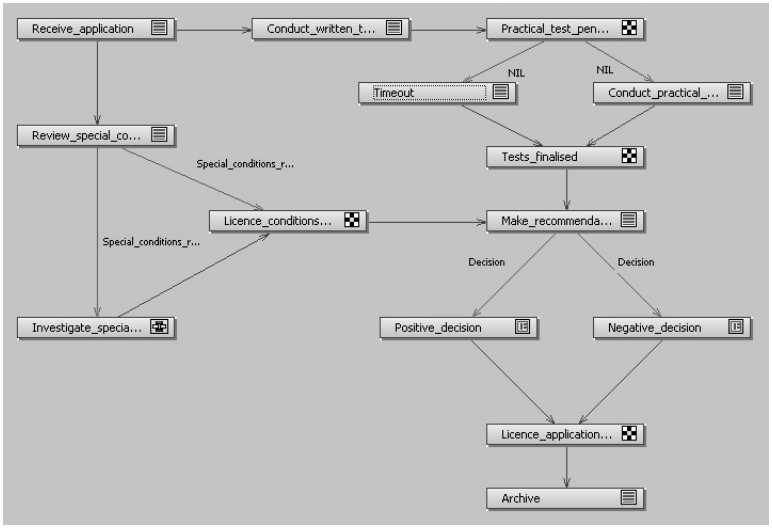


Figure 3.14
Case-type model of the license application process.

Data objects have a preeminent role during the execution of a case and, unlike traditional workflows, determine the manner in which case execution occurs. Data objects are attached to a process and correspond to a piece of information that may or may not be present. Where a data object is present, it has a value recorded for it. Data objects can be either *free*, meaning they can be amended at any time during case execution and are not explicitly linked to a specific task in the process, or *mandatory* and/or *restricted*, in which case they are linked to one or more specific tasks. A mandatory data object means that its value must be recorded before the task to which it is attached can complete. It may be provided ahead of the time that the task is executed or during its execution, but either way it must be available at task completion. In contrast, a restricted data object can only have its value recorded during the execution of the task(s) with which it is associated.

The resources that undertake the tasks in a case are termed *actors*. They are organized in terms of *roles* that group actors with similar responsibilities or capabilities. An actor may be a member of more than one role. Roles are specific to a given process and can be linked together via *is_a* relationships to illustrate the relationship between them. Actors that are members of a superior role in the overall hierarchy are automatically members of any of its subordinate roles.

Three types of roles can be specified for each process and activity:

- the *execute* role indicates which actors can execute a task or start a process instance.
- the *redo* role indicates which actors can undo a task reverting the state of the case to that before the task executed. Undoing a task also undoes the effects of all subsequent

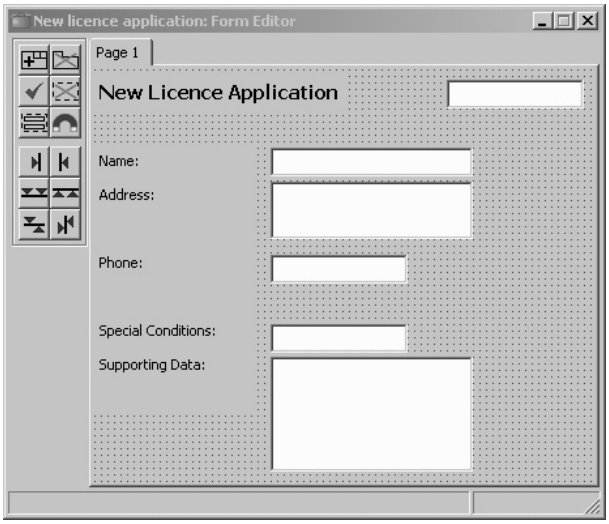


Figure 3.15
Form definition in BPMone.

activities. (Of course it is impossible to undo externally visible actions, e.g., a payment. In such cases, a compensating action is needed.)

- the *skip* role indicates which actors can skip over the execution of a task.

There is a significant difference in the way that work is distributed when using a case handling approach. Unlike traditional workflow systems, which rely on the use of an in-tray or worklist client to indicate which tasks an actor must undertake, case handling relies on the use of case queries that are more flexible in form and allow an actor to search for any tasks that may need attention in order to complete a case. It does not require that these tasks be distributed to the actor in order to undertake them.

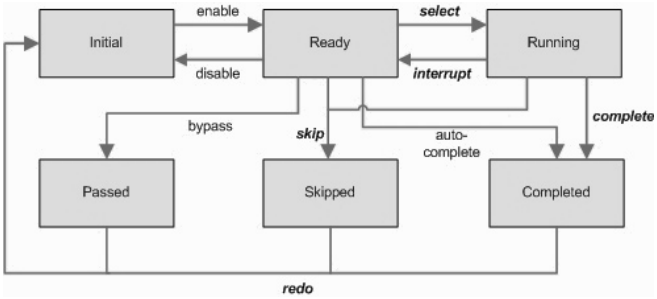


Figure 3.16
Task lifecycle in BPMone.

The lifecycle for each step in BPMone is illustrated in figure 3.16. User-initiated actions via the worklist are shown in bold. Once created at case initialization, steps become enabled when any precursing data constraints have been met. Once in a *ready* state, they can be commenced (and enter a *running* state) when selected by a user. The user can interrupt them at any time by returning to the worklist (causing a transition to a running state) or can complete them, causing a transition to a *completed* state. A *ready* or *running* step can be skipped by a user at any time, triggering a change to a *skipped* state. Where configured accordingly in the process model, a *ready* step can transition automatically to a *passed* or *completed* state where qualifying conditions for this event are met. Finally, a user can trigger the repetition of a step in the *passed*, *skipped*, or *completed* state via an explicit redo request in the worklist handler. Each of these states has a distinct annotation in the BPMone worklist.

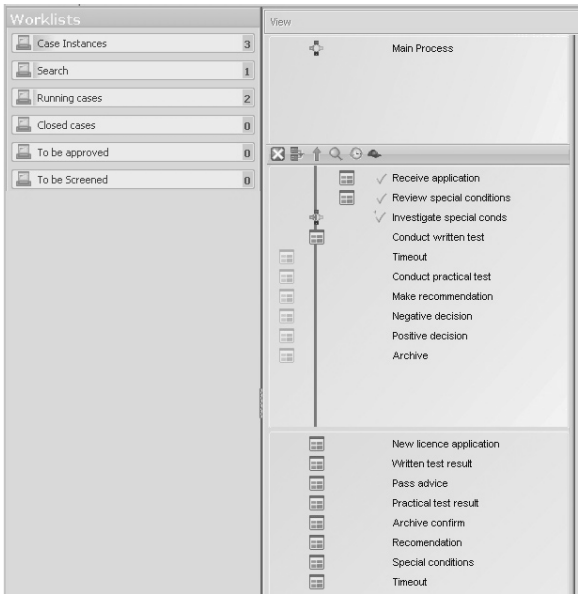


Figure 3.17
Worklist operation in BPMone.

Figure 3.17 shows the worklist interface for a user. On the left are the standard work queues into which cases can be retrieved. The right-hand side shows an instance of the license application process that is currently in progress. The top part of this frame indicates that the "Main Process" (i.e., the top-level process) is active at present. The middle frame shows the current state of the case. The bottom frame shows the forms that make up the process. The vertical blue line in the middle frame is known as the *wavefront*, and it indicates which steps are currently active. Anything to the right of the wavefront has been

completed or is no longer active (i.e., the *Receive application* and *Review special conditions* steps are complete as indicated by the green tick). The *Investigate special conds* step has also been completed but is currently back on the wavefront as a user has requested it be redone. The *Conduct written* test step is also currently active. Anything to the left of the wavefront is not yet active. However, unlike traditional workflow tools, in BPMone, a user can elect to *execute*, *skip*, or *redo* any step at any time. There is no prescribed ordering for the process, only a suggested sequence based on interdependencies between steps. Where a step is done outside of the recommended sequence and changes are made to associated data elements, any subsequent steps depending on the step are also redone.

The case handling paradigm makes a significant contribution to the BPM landscape and provides a valuable alternative approach to executing business processes. It especially highlights the need for flexible business processes that are better able to accommodate special operational requirements and deviations that may arise during execution. This is a critical issue in today's turbulent business environment. The following section examines these needs in more detail.

3.5 Flexibility

A desirable characteristic for business processes is that they are resilient to changes that may arise in the operational environment. The increased prevalence of automated business process support and the volatile nature of the environments in which they are deployed mean that it is increasingly likely that during their operation, business processes may encounter unusual and/or unanticipated situations that need to be catered for. The ability of a business process to successfully respond to unanticipated events or changes is termed *flexibility* or *agility*.

There are a variety of ways in which business processes can respond to these situations, and these are enumerated in the *Taxonomy of Flexibility* [121], a broad classification of approaches to facilitating process flexibility derived from a comprehensive survey of flexibility support in BPMSs and the BPM literature.

This taxonomy recognizes five distinct approaches to incorporating flexibility in a process, and the relationship between them is shown in figure 3.18. These approaches are as follows:

- *Flexibility by design* involves the explicit incorporation of flexibility mechanisms into a process at design-time using available process modeling constructs such as splits and joins;
- *Flexibility by deviation* involves supporting the ability for individual process instances to temporarily deviate from the prescribed process model in order to accommodate change that may be encountered in the operating environment at runtime;

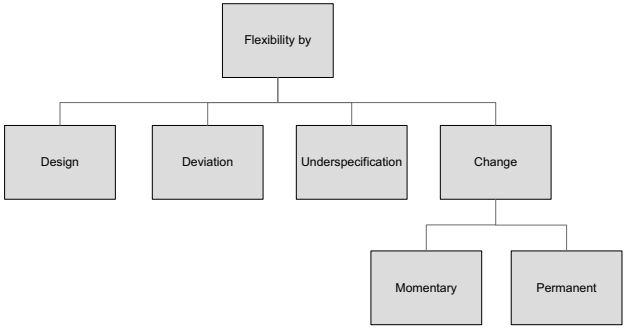


Figure 3.18
Taxonomy of flexibility types.

- *Flexibility by underspecification* is the ability to deliberately underspecify parts of a process model at design-time in anticipation of the fact that the required execution details will become known at some future time. When a process instance encounters an underspecified portion of a process at runtime, the missing part can be instantiated in accordance with the required operational outcomes and then executed; and
- *Flexibility by change* where changes in the operational environment are catered for by actually amending the process model. There are two approaches to this:
  - — *Flexibility by momentary change* involves changing the process model corresponding to a specific process instance. Any other process instances are unaffected by the change; and
  - — *Flexibility by permanent change* involves changing the process model for all process instances. This may involve migrating any other currently executing instances from the old to the new process model.

Contemporary BPM modeling languages and BPMSs offer varying degrees of support for process flexibility as illustrated in table 3.1. In general, flexibility by design is well supported by most offerings reflecting the common preconception that flexibility can be foreseen and designed into a process. Less well supported are other approaches to flexibility that involve (1) relaxing the strict approaches to process execution at runtime and allowing for deviation from the prescribed execution sequence, (2) allowing process designs to be deliberately underspecified with the expectation that any shortfalls in the precision or clarity of a process can be rectified before or at runtime, or (3) allowing processes to be changed at runtime, for either a single execution instance or all execution instances.

Although BPMone provides a modicum of support for flexibility by deviation as a consequence of the case handling approach it adopts to process execution and Oracle BPEL supports late binding, in general terms, there is minimal support for enabling flexible process behaviors among contemporary BPM languages and systems. However, there are a

Table 3.1

Flexibility type support for BPMN, BPMone. Oracle BPEL, YAWL with worklets, ADEPT1 and Declare : + indicates full support, +/- partial support, and – no support (results for ADEPT1 and Declare from [121]).

Flexibility Type	Realization Option	BPMN	BPMone	Oracle BPEL	YAWL + worklets	ADEPT1	Declare
Flexibility by Design	Parallelism	+	+	+	+	+	+
	Choice	+	+	+	+	+	+
	Iteration	+	+	+	+	+	+
	Interleaving	+	+	+	+	–	+
	Multiple instances	+	+	+	+	–	+
	Cancellation	+	–	+	+	–	+
Flexibility by Deviation	*Imperative languages*						
	Undo	–	+	–	–	–	N/A
	Redo	–	+	–	–	–	N/A
	Skip	+	+	+	–	–	N/A
	Create additional instance	–	+/–	–	–	–	N/A
	Invoke task	+	+	+	–	–	N/A
	Declarative languages						
	Violation of constraints	N/A	N/A	N/A	N/A	N/A	+
Flexibility by Underspecification	*Placeholder enactment*						
	Late binding	+	–	+	+	–	–
	Late modeling	–	–	–	+	–	–
	Moment of realization						
	Static, before placeholder	–	–	–	–	–	–
	Dynamic, before placeholder	–	–	–	–	–	–
	Static, at placeholder	–	–	–	–	–	–
	Dynamic, at placeholder	+	–	+	+	–	–
Flexibility by Change	*Effect of Change*						
	Momentary change	–	–	–	–	+	+
	Evolutionary change	–	–	–		–	+
	Moment of allowed change						
	Entry time	–	–	–	–	+	+
	On–the–fly	–	–	–	–	+	+
	Migration strategies for evolutionary change						
	Forward recovery	–	–	–	–	–	–
	Backward recovery	–	–	–	–	–	–
	Proceed	–	–	–	–	–	+
	Transfer	–	–	–	–	–	+

number of approaches utilized within current BPM tools that go some way toward supporting expected and unexpected events that may arise during process execution. Dynamic workflow provides a means of extending processes at runtime to deal with situations not anticipated at design-time. Exception handling provides a means of dealing with undesirable events that arise during process execution. Declarative workflow offers an alternative means of specifying and executing processes that focus on what needs to be achieved rather than how it should be done. Each of these approaches is described in the remainder of this chapter.

3.5.1 Dynamic workflow

Dynamic workflow provides a means of extending and adapting design-time process definitions at runtime to cater to unanticipated changes in either the required process behavior or unexpected events arising in the execution environment. There are two possible approaches to dealing with such scenarios: (1) deliberately underspecify the design-time process model and allow it to be extended when the required process behavior becomes

known or unexpected events occur, or (2) provide a range of mechanisms within the workflow environment to allow an executing process to be dynamically changed at runtime to deal with evolving requirements and events occurring in the operational environment that the process must cater to.

An example of *flexibility by underspecification* is provided by the Worklet service in the open-source YAWL workflow engine. The Worklet service provides a repository for a range of *worklets,* each of which corresponds to a group of activities intended to deal with a particular processing requirement. A worklet takes the form of a complete YAWL process. For any given YAWL process, it is permissible to indicate that a given task will be fulfilled by a worklet at runtime. This removes the immediate need to specify all implementation details for the task at design-time and instead passes responsibility for this to the Worklet service. When the task is triggered at runtime, the Worklet service matches the task up with an appropriate worklet based on the current state of the process and its associated data elements. It utilizes a "Ripple Down Rule" (RDR) tree for this purpose. In the event that a suitable worklet cannot be located, it is possible to dynamically add either a new case for the RDR tree or even a complete new worklet. An example of such an RDR tree is shown in figure 3.19 for an order returns processing task. Depending on whether the goods were damaged, the number of returns for the customers this year, the seniority of the staff member handling the return, the value of the return, whether the customer is a staff member, and the status of the customer, one of several distinct worklets is selected as the means by which the task is operationalized at runtime.

The ADEPT/ADEPT2/AristaFlow lineage of workflow engines chart the cutting edge of research into *flexibility by change*. Originally conceived as a research prototype, the AristaFlow BPM Suite is now available as a fully fledged commercial offering. It provides users and process administrators with the ability to intervene in a process where an unexpected situation has arisen. In order to resolve the situation within the context of the process, a range of facilities are provided to allow the process to be dynamically changed to cater to a wide range of exceptions, including the addition or deletion of elements (which may be activities, splits, joins, loops, branches, etc.) in the process, and the addition of, deletion of, or changes to the use of data elements within a process. In order to ensure that the operational integrity of the process is maintained, any changes are subject to a consistency check to ensure they do not invalidate the correctness of the process. Any such changes can be limited to a single process instance or may be propagated to a range of process instances. There is also provision for structural changes to an overall process model and the propagation of these changes to executing process instances. Figure 3.20 illustrates the addition of a *review findings* task to a medical process.

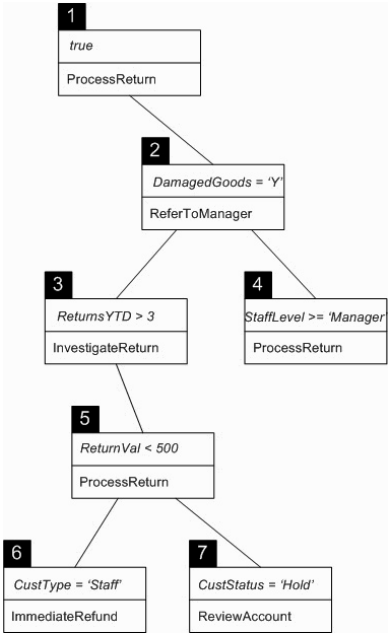


Figure 3.19
RDR tree selection of worklet.

3.5.2 Exception handling

Exception handling is a technique that has been widely used for many years in software development as a means of dealing with the range of scenarios that may occasionally be encountered during the execution of a program that are not specifically catered to as part of its processing logic. The basic premise of exception handling is that a generic strategy is defined that can best deal with the effects of a specific type of error (or indeed all types of unexpected errors) that might arise during execution of a business process. In this section, we will consider two distinct approaches to exception handling that are commonly used in BPM systems.

Exception handlers

Exception handlers have been utilized in the software engineering field for many years as a means of responding to a broad range of unanticipated program errors that might arise at runtime. They typically take the form of a procedure that is associated with a given process or a block within a process that describes the action to be taken when a nominated event occurs. This event can be system-generated (e.g., hardware fault, divide by zero error) or application-generated (e.g., remote service call returns an error). Usually a given handler

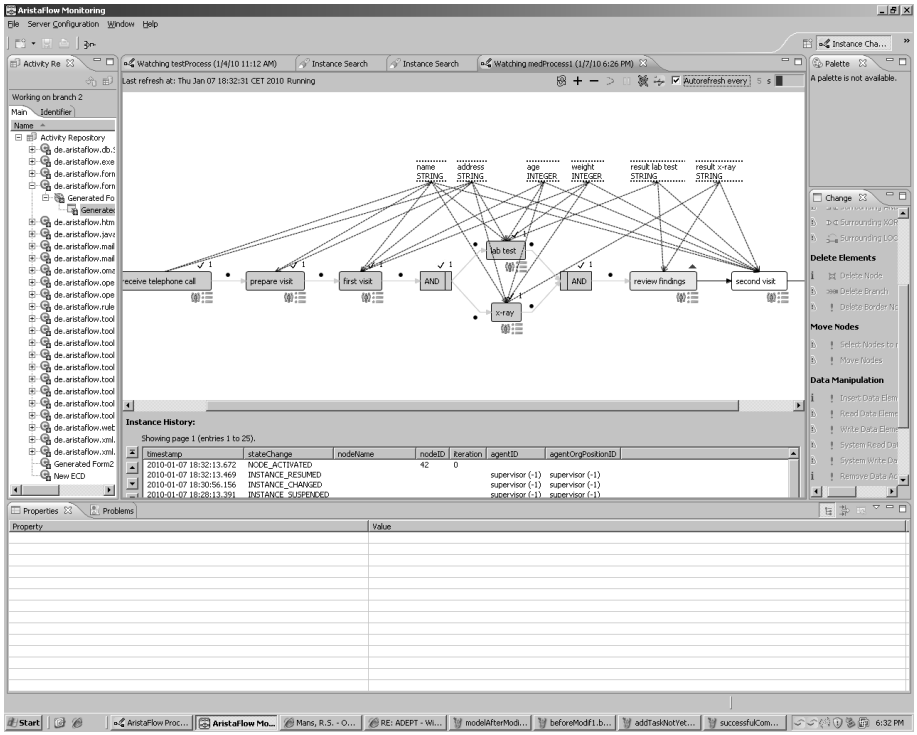


Figure 3.20
Support for *flexibility by change* in AristaFlow.

has a specific exception that it responds to and aims to resolve; however, it is possible to define blanket exception handlers for all detected errors.

When the nominated exception is detected, the execution of the part of the process to which the exception is attached ceases, and the thread of control is passed to the exception handler. The exception handler takes any necessary steps that it can to resolve the exception and then at its conclusion can do one of three things: (1) cease execution and pass the thread of control back to the place in the process instance from which it was invoked, (2) cease execution and trigger no further action, or (3) escalate the detected exception to a higher level by triggering (often termed "throwing") another exception handler possibly in a surrounding program block.

Oracle BPEL provides a range of exception handling capabilities in the form of fault handlers, event handlers, and compensation handlers. The use of these features provides a wide array of options for dealing with undesirable events that may occur during the execution of a process. Depending on the handling option being utilized, these events may be expected or unexpected, the distinction being whether the process definition identifies

the potential for the event to arise and how to deal with it or not. In all cases, the use of these features requires the relevant handling behavior to be associated with a specific scope within a process definition.

A *fault handler* reverses the effects of partially complete or unsuccessful work in the scope in which the fault is detected. BPEL identifies two types of faults: (1) *business* faults, which are application-specific and generally relate to *information* being processed; and (2) *runtime* faults, which arise as a consequence of problems in the operating environment. Business faults occur when a process recognizes that an error has occurred and initiates a *throw* activity or when an *invoke* activity generates a fault during its operation. In contrast, runtime faults are triggered by the system when it detects a problem during the execution of a BPEL process (e.g., logic errors, messaging faults). A fault handler takes the form of a sequence of activities or a process that is associated with a specific error type in a given scope. Once a fault is detected, control is passed to the relevant handler in the scope in order to deal with the detected exception. At its conclusion, it may terminate any further execution of the process, resume execution in the scope in which the exception was detected, or escalate the fault by propagating it to the surrounding or parent scope.

In contrast to a fault hander, a *compensation handler* undoes the effect(s) of a successfully completed scope. It can also allow for the undoing of a multi-part or distributed transaction that has partially completed execution. A compensation handler takes the form of a sequence of activities or a process that is associated with a specific scope. It is invoked via the *compensate* activity and can only be triggered in the fault handler or compensation handler of the scope that immediately encloses the scope being compensated. Once the requested compensation has been completed, the thread of control passes to the activity immediately following the compensate activity.

An *event handler* provides a means for a scope to respond to specific events that may arise during its execution that need to be dealt with. In the main, these are timeouts or the receipts of specific types of messages. Similar to fault handlers, event handlers correspond to sequences of activities or processes associated with a scope and have an *OnAlarm* or *OnMessage* event that defines when they are to be invoked. At their conclusion, they can either terminate the process or pass the thread of control back to the scope in which they were triggered. In contrast to fault handlers, which address error situations, event handlers are assumed to form part of the normal processing sequence where some form of asynchronous event needs to be dealt with.

Dynamic process extension

An alternate means of dealing with unanticipated events during process execution is provided by dynamic process extension. This involves extending the process with the necessary tasks to deal with the effects of the detected event at runtime. Typically this approach to exception handling involves manual intervention during process execution when an exception is detected in order to amend the process instance in a way that addresses the issue.

Although this is a novel concept in contemporary BPM offerings, the open-source YAWL workflow engine provides the *Exlet* feature as part of its Worklet service that allows for the runtime adaptation of processes to cater for unanticipated or undesirable events. In much the same way that worklets provide a repertoire of alternate implementation behaviors for an activity, a set of exlets provides the ability to specify a repertoire of exception handling behaviors that can be utilized throughout a process definition. An exlet can be bound to either an individual activity or a process instance within an operational BPM environment. Each exlet has a unique event that causes it to execute. Supported events include pre/postconstraint violations, time outs, external triggers, abort requests, and unavailable resources. An exlet is defined in terms of a sequence of primitives that describe how an unanticipated event should be handled when detected during process execution. In order to provide a rich array of constructs with which to define exception handling strategies, the standard language primitives in YAWL are extended with sixteen additional language elements specifically aimed at exception handling. These are illustrated in figure 3.21.

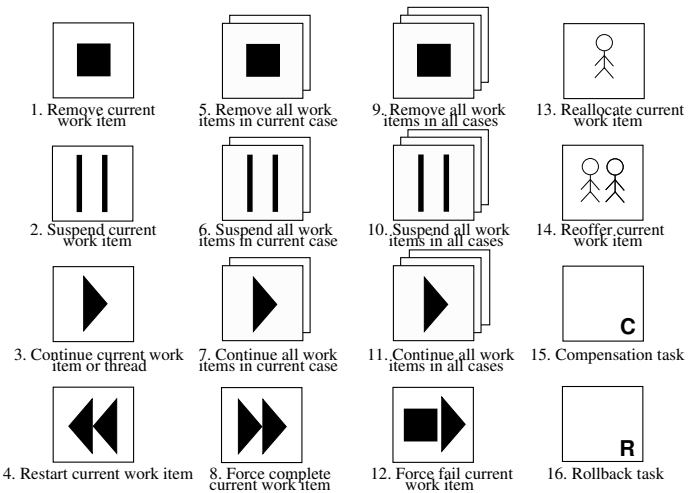


Figure 3.21

Exlet exception handling primitives (from [112]).

An exlet operates in much the same way as a worklet. When the trigger event for a specific exlet is detected during process execution, the relevant exlet is initiated and attempts to resolve the detected event. Depending on the primitives defined for the exlet, further execution of "normal" process related activities may continue or it may be suspended. At its conclusion, the exlet can pass control back to the process, cause the process instance that detected it to terminate, or even cause all related process instances to terminate. Figure 3.22 illustrates the use of exlets to handle exception events detected in the context of a *pack order* task in an order fulfillment process. Three distinct exlets are defined for this activity:

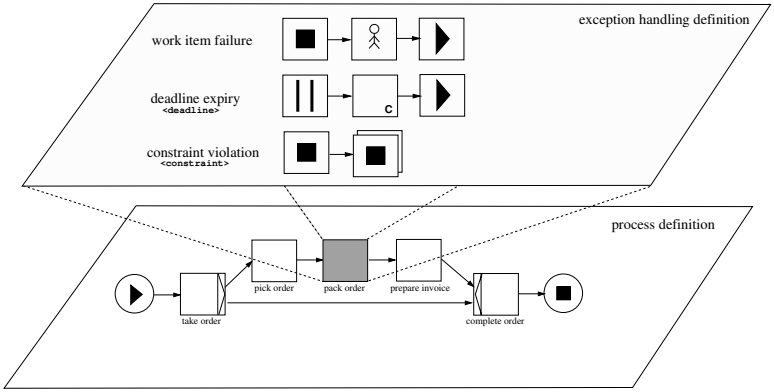


Figure 3.22
Exlet event handing strategy for order process (from [112]).

(1) in the event of the work item associated with the activity failing, it is stopped, manually reassigned to another resource, and continued; (2) in the event of a deadline for the activity being exceeded, it is paused, any necessary compensation is undertaken (e.g., advising the customer of the delay), and then it is continued; and (3) in the event of a constraint associated with the task (e.g., outstanding customer credit) being exceeded, the task is stopped and all other tasks in the current process instance are also stopped.

3.5.3 Declarative workflow

Today's process-aware information systems tend to either support business processes *or* provide flexibility. Classical WFM systems offer good process support as long as the processes are structured and do not require much flexibility. Information systems that allow for flexibility have a tendency to provide little process-related support. When enabling business processes using IT, there is a difficult trade-off to be made. On the one hand, there is a desire to control processes and avoid incorrect or undesirable executions of these processes. On the other hand, users want flexible processes that do not constrain them in their actions. This apparent paradox has limited the application of WFM/BPM technology thus far because, as indicated by many authors, classical systems are too restrictive and have problems when dealing with change. The case-handling paradigm used by BPMone and the worklets/exlets of YAWL aim to address this problem. Another approach to balance between support and flexibility is provided by *Declare* [17].

Traditional approaches tend to use procedural process models to explicitly (i.e., step-by-step) specify the execution procedure. Declarative approaches are based on *constraints* (i.e., anything is possible as long as it is not explicitly forbidden). Constraint-based models,

therefore, implicitly specify the execution procedure by means of constraints: any execution that does not violate constraints is possible.

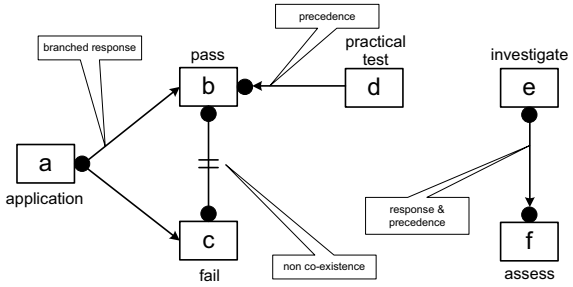


Figure 3.23

Declare specification consisting of five constraints: two precedence constraints, one non-co-existence constraint, one response constraint, and one branched response constraint.

Declare is an example of a declarative constraint-based language (in fact a family of languages). Moreover, Declare is supported by fully functional BPM system [17, 85]. Declare uses a graphical notation and semantics based on Linear Temporal Logic (LTL) [34]. Figure 3.23 shows a Declare specification consisting of four constraints. The construct connecting activities b and c is a so-called *non-co-existence constraint*. In terms of LTL, this constraint means "$!((\Diamond b) \wedge (\Diamond c))$"; $\Diamond b$ and $\Diamond c$ cannot both be true (i.e., it cannot be the case that both b and c happen for the same case). There are two *precedence constraint*s. The semantics of the precedence constraint connecting d to b can also be expressed in terms of LTL: "$(!b)\ W\ d$" (i.e., b should not happen before d has happened). Because the weak until (W) is used in "$(!b)\ W\ d$," execution sequences without any b and d activities also satisfy the constraint. The constraint connecting a to b and c is a so-called branched constraint involving three activities. This *response constraint* states that every occurrence of a should eventually be followed by b or c: "$\Box(a \Rightarrow (\Diamond(b \vee c)))$." Note that the latter constraint allows for $\langle a,a,a,b,c,a,a,b\rangle$ but not $\langle a,b,c,a\rangle$. The constraint connecting e to f is a combination of a response and a precedence constraint. The response part states that "$\Box(e \Rightarrow (\Diamond f))$" (i.e., any occurrence of e should eventually be followed by f). The precedence part states that "$(!f)\ W\ e$" (i.e., f should not happen before e has happened). Example traces that satisfy all four constraints are $\langle a,a,d,b\rangle$, $\langle a,a,c,c,c\rangle$, $\langle e,e,f,f\rangle$, $\langle d,d,e,a,e,b,f,b,f\rangle$, and so on.

Procedural languages only allow for activities that are explicitly triggered through control-flow (token semantics). As indicated before, in a declarative language like Declare, "*everything is possible unless explicitly forbidden*." This is illustrated by figure 3.23, which allows for a wide range of behaviors difficult to express in procedural languages.

The Declare language is supported by the Declare system, which is much more flexible than traditional procedural WFM systems [17]. Moreover, it is possible to learn Declare

models by analyzing event logs [85]. The graphical constraint language is also suitable for conformance checking (one of the types of process mining introduced before). Given an event log, it is possible to check all constraints. Consider, for example, figure 3.23. Given an event log, one can compute for each constraint the proportion of cases that respects this constraint [17, 85]. In the case of conformance checking, complex time-based constraints may be used (e.g., after every occurrence of activity *a* for a gold customer, activity *b* or *c* should happen within 24 hours).

3.6 Current challenges

This chapter has outlined the development of the BPMS field and the role it has played in automating the operation of business processes. There have been significant developments over the past 30 years, and the available technology now provides an effective means of enacting business processes in an organizational context. Nevertheless, the BPM domain remains in a continual state of flux as it continues to evolve. In this section, we examine some of the current challenges that exist in the BPM arena.

Absence of widely adopted BPM standards

One of the motivations for the establishment of the Workflow Reference Model by the WfMC in 1995 was to establish a common vocabulary for the workflow domain. By doing so, it was anticipated that the foundation would be laid for interchange of process models between the wide variety of technologies falling under the workflow umbrella. In essence, the aim of the WfMC was to promote the use of workflow technology in a consistent way regardless of the underlying formalisms on which it was based. To some degree the effort was successful in establishing a basic vocabulary for common workflow terms, which is relatively widely used in the BPM community. However, attempts at establishing a common reference model for workflow definition and execution were largely ignored by workflow developers particularly because the proposed workflow execution model was not especially powerful, and many did not see any benefit in migrating their proprietary modeling formalisms to a common technique. More recently, BPEL achieved relatively widespread support among vendors as a common format for specifying executable business processes based on web services; however, it suffered from the difficulty that it was only informally defined, leading to varying interpretations of its constructs and operation. Perhaps the most successful initiative in this area to date has been the continued evolution of BPMN. Initially positioned as a modeling formalism and only informally defined, it has now matured into a fully fledged business process modeling and execution language based on a comprehensive meta-model together with an associated graphical modeling notation

and an execution semantics defining how BPMN processes should be enacted. It is increasingly being seen as the de facto standard for business process definition and has been adopted by a wide range of vendors.

Disparity between modeling and enactment tools

Traditionally in the BPM domain, distinct formalisms have been used for the modeling of business processes at the conceptual and executable levels. The modeling strategies used at the conceptual level have concentrated on providing a wide range of modeling artifacts without any immediate consideration of how they will be operationalized. Examples of such techniques include EPCs, UML activity diagrams, and BPMN. Techniques used at the executable level focus on the ability to enact the various elements of a process model and consequently are designed with determinism in mind rather than conceptual power. Where a business process is defined in terms of a conceptual modeling notation such as EPCs or UML activity diagrams, there is a consequent mapping activity that is required into an executable format before the process can actually be enacted. The semantic disparity between conceptual and executable modeling notations means that there is generally some form of (manual) remodeling of a process required during this mapping activity, and consequently there is the potential for information loss and misinterpretation of the candidate business process. Two distinct approaches have been proposed to resolving this issue. The first focuses on the development of execution semantics for individual conceptual modeling notations that allows them to be directly enacted without ambiguity. Perhaps the most successful initiative in this area is BPMN 2.0.2, which is now the basis of a number of commercial offerings such as Oracle BPM Suite, which provides direct execution support for process models specified using BPMN.

Incorporating flexibility in business processes

The modern business environment is in a continual state of flux, and it is a desirable property of a business process that it is able to accommodate the various types of changes that might arise during its operational lifetime. Workflow technology has faced significant criticism in recent years in regard to its inflexibility and inability to adapt to changing operational circumstances. Consequently, there is increased focus on incorporating various types of flexibility support into business processes as a means of improving their overall resilience. As indicated in section 3.5, four distinct approaches have been identified to incorporating change support in processes: (1) *flexibility by design,* (2) *flexibility by deviation,* (3) *flexibility by underspecification*, and (4) *flexibility by change*. Individual BPMS offerings are starting to provide various types of flexibility support in specific areas (e.g., BPMone provides a number of deviation facilities); however, widespread flexibility support among BPMS offerings has not yet emerged).

Supporting commercial-scale business processes

The potential benefits of BPM come to the fore when automating large-scale, complex business processes that operate on a "whole-of-organization" basis or between several organizational participants in a business process. Such processes are so broad in form that they do not fall under the auspices of any one individual but instead have specific sections that are managed by distinct groups of resources. It is possible that all of the expertise required to undertake a given process does not lie within the organization and that the business process also needs to integrate external resources and services in order to achieve corporate goals, or indeed that a business process operates between several organizations and coordinates their interactions in an overall business initiative. When automating business processes of this scale, a new range of issues arise that are not evident in smaller processes that involve less complex set of activities or operate over a shorter time horizon, such as handling the evolution of individual sections of the process, versioning of process instances, coordinating resource assignment, managing workload, amending/outsourcing/insourcing task implementation, and monitoring process operation to ensure expected service levels are achieved. The criticality of business processes of this scale means that these issues must be dealt with on a dynamic basis and cannot necessitate the termination or restart of a process. The introduction of service orientation as a means of constructing business processes offers some initial support for dealing with some of these issues; however, there is still significant work required in this area.

Empowering users in the business process

There has been a great deal of work conducted on the optimization of business processes from a global standpoint; however, in many situations, it is actually the individual users who undertake specific activities within a process that are best positioned to identify potential opportunities for improvement. Consequently, there is significant potential in providing users with an insight into the operation of a process instance in which they are involved and the impact that their individual work activities have on it. This information is particularly useful when coupled with a range of alternative options (e.g., start a work item, delegate a work item, escalate a work item, etc.) from which a user can select how they will interact with a process in the most beneficial way. Providing suitable intelligence to users about process performance may take a number of forms. As illustrated in section 2.5, one possible approach involves the use of *process mining*, which seeks to provide users with decision-making information based on the analysis of preceding executions of a process. Although there are a variety of different analysis techniques with which to do this, the most important consideration is ensuring that the necessary information is presented to the user in a meaningful way at the exact time that they require decision support. This approach can be especially powerful when linked with richer worklist visualization facilities, which provide workflow users with a much broader range of information about outstanding

work items when making a decision as to which work item to execute next or which work items need prioritizing. The YAWL workflow system has been successfully coupled with the ProM process mining suite as a demonstration of how this may be achieved, and it is anticipated that similar capabilities will find their way into commercial offerings in due course.

Leveraging the knowledge contained in business processes

Achieving best practice in the operation of a given business process is one of the overarching aims of BPM deployment. In doing so, organizations seek to optimize the function of a business process in an effort to maximize both its effectiveness and efficiency. Reference models have long been advocated as a means of distributing knowledge in regard to optimal process designs for a given domain. These typically present standardized processes for common business activities in a form that facilitates their reuse by other organizations that do not have prior experience in their deployment and serve as a guideline for the form that a candidate business process should take. Although the rationale for the use of reference models is sound, in practice, however, it is often found that the specific context in which a reference model can be applied is extremely limited, and large parts of the "standard model" are not generally applicable to organizations seeking to use them. Configurable reference models have been proposed as a solution to this difficulty and offer a means of describing a standard business process along with the various configuration alternatives that might exist for it, thus allowing an organization to tailor it to their specific requirements. The concept has been successfully demonstrated for the EPC and YAWL modeling languages, but in order for it to achieve more widespread usage, it needs to be extended to the wide range of mainstream business process modeling languages, and there needs to be a comprehensive library of standard business processes that is freely available to organizations seeking to use them.

3.7 Further reading

- Full details of the Workflow Reference Model can be found in [143]. A wealth of related workflow material is also available from the website `www.wfmc.org`.
- Widely recognized texts on workflow technology have been written by Van der Aalst and Van Hee [10], Leymann and Roller [79], and Jablonski and Bussler [68].
- The definitive work on Process-Aware Information Systems (PAIS), the broader class of technologies of which BPMSs are a part, is that by Dumas et al. [42].
- The definition for XPDL 2.2 is contained in [145] available from the WfMC website.
- The current version of BPEL (2.0) is defined in [93]. The related BPEL4People and WS-HumanTask specifications are both now at version 1.1, full details are available

in [95] and [94] respectively. Comprehensive documentation on Oracle BPEL is available from the Oracle website at www.oracle.com/technetwork/middleware/bpel/overview/index.html.

- A comprehensive overview of case handling can be found in [21]. Further information on the BPMone product is available from Perceptive Software's website at www.perceptivesoftware.com.
- An overview of the issues associated with process flexibility can be found in [23] and [104]. The worklets and exlets aspects of the YAWL system (including the use of Ripple-Down Rules) are comprehensively documented in [22]. Further details on YAWL can be found at www.yawlfoundation.org and in [64].
- The chronology of the ADEPT/ADEPT2/AristaFlow system is documented in [37]. Further details on the software are available from www.aristaflow.com.
- An introduction to the ProM initiative can be found in [18]. The software and comprehensive technical information about the offering are available at www.processmining.org.
- The widely recognized standard text on process mining is [3].
- Details on the constraint-based language Declare, the Declare system, and the use of process logs to develop Declare models can be found in [17] and [85].
- The concept of configurable reference model languages is introduced in [109], and details of the implementation of a configurable workflow model (in the context of the YAWL system) is presented in [58]. A comprehensive treatment of process model variability and configuration strategies in contained in [108].
- [103] provides a comprehensive treatment of flexibility and change management issues in the context of BPMSs.