Chapter 9
Process Automation

Besides black art, there is only automation and mechanization.
Federico García Lorca (1898–1936)

This chapter deals with process automation. First, we will briefly explain what an automated business process is, after which we will focus on a specific kind of technology that is particularly suitable to achieve process automation, i.e. Business Process Management Systems (BPMSs). We will explain the features and advantages of these systems, present the different types of BPMS, and discuss some of the challenges that are involved with introducing a BPMS in an organization. Finally, we will discuss what changes are required to a *business-oriented* process model like the ones seen so far, to make it *executable* and run in a BPMS.

9.1 Automating Business Processes

Process automation is a subject that may be approached from different angles. In a broad sense, it may refer to the intent to automate *any* conceivable part of procedural work that is contained within a business process, from *simple* operations that are part of a *single* process activity up to the automated coordination of *entire*, complex processes.

Take, for example, the order fulfillment process that we modeled in Chap. 3. Automating such a process may imply that every time the seller receives a purchase order, this is automatically dispatched to the ERP systems of the warehouse and distribution department, where the availability of the product is checked against the warehouse database. If the product is not in stock, the relevant suppliers are automatically contacted, e.g. via a Web service interface, to manufacture the product. Otherwise, instructions are sent to a warehouse worker, e.g. using an electronic form, to manually retrieve the product from the warehouse. Subsequently, an order clerk from sales receives a notification that a new order needs to be confirmed, e.g. via email. That clerk would then log into the purchase order tracking system within sales, see the order electronically, and confirm it by pressing a button, and so on.

In this example the dispatching of the purchase order, the automated check of the product's availability, or the automated web messages are all manifestations of process automation in its broadest sense: they automate a particular aspect of a process. In this context, we will refer to an *automated business process*, also known as *workflow*, as a process that is automated in whole or in part by a software system, which passes information from one participant to another for action, according to the temporal and logical dependencies set in the underlying process model. Let us now consider systems that work with automated business processes.

9.1.1 Business Process Management Systems

A specific kind of process automation, which we are particularly concerned with in this book, exploits knowledge about how different process activities *relate* to one another. In other words, the type of information systems that we consider are *process-aware*. The main category of process-aware information systems that we will discuss are the so-called *Business Process Management Systems* (BPMSs). While there are other types of process-aware system, such as Customer Relationship Management (CRM) systems and Enterprise Resource Planning (ERP) systems, the special feature of BPMSs is that they exploit an explicit description of a business process, in the form of a *process model*, to coordinate that process. In that sense, a BPMS can be tailored to specific processes of any kind.

The purpose of a BPMS is to coordinate an automated business process in such a way that all work is done at the right time by the right resource. To explain how a BPMS accomplishes that, it is useful to see that a BPMS is in some way similar to a *Database Management System* (DBMS). A DBMS is a standard, *off-the-shelf* software package offered by many vendors in many different flavors, such as Microsoft SQL Server, IBM DB2 or Oracle Database. With a DBMS it is possible to capture company-specific data in a structured way, without ever having to consider how the exact retrieval and storage of the involved data takes place. These tasks are taken care of by standard facilities of the system. Of course, at some point it is necessary to configure the DBMS, fill it with data, and it may also be necessary to periodically adapt the system and its content to actual demands.

In a similar manner, a BPMS is also a standard type of software system. Vendors offer different BPMSs with a varying set of features, spanning the whole process lifecycle: from simple systems only catering for the design and automation of business processes, to more complex systems also involving process intelligence functionality (e.g. advanced monitoring and process mining), complex event processing, SOA functionality, and integration with third-party applications and social networks. Despite the variety of functionality a BPMS can offer, the core feature of such a software system resides in the automation of business processes. With a BPMS it becomes feasible to support the execution of a specific business process using the standard facilities offered by the system. However, it is essential that a business process is captured in such a way that the BPMS can deal with it, i.e. that

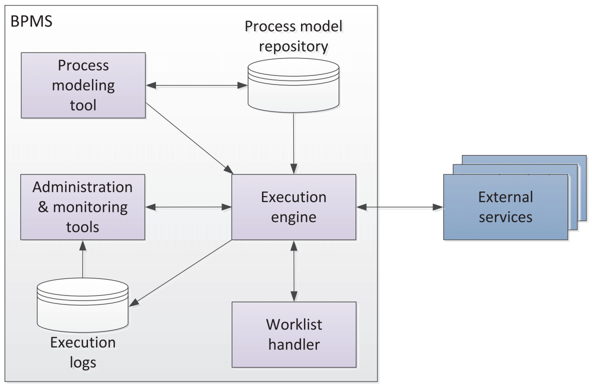


Fig. 9.1 The architecture of a BPMS

the BPMS can support its execution. From the moment a process is captured in a format that the BPMS can work with, it is important to keep that description of the business process up-to-date so that the process is supported properly over time.

In the past, mainly before the emergence of BPMSs, there existed a large number of tools focused on process automation, which did not encompass process intelligence functionality and which had relatively minimal support for process modeling. These tools were known under the name of *Workflow Management Systems* (WfMSs). Over time, many of these tools evolved towards BPMSs. Additionally, a plethora of stand-alone tools exist that cover a single feature of advanced BPMSs, such as stand-alone process modeling tools, process simulation tools and process analytics tools. All these tools provide value in supporting various parts of the BPM lifecycle, but do not generally support process automation. For this reason, they will not be the focus of this chapter.

9.1.2 Architecture of a BPMS

Figure 9.1 shows the main components of a BPMS, namely the execution engine, the process modeling tool, the worklist handler, and the administration and monitoring tools. The execution engine may interact with external services.

Execution Engine Central to the BPMS is the *execution engine*. The engine provides different functionalities including: (i) the ability to create executable process instances (also called cases); (ii) the ability to distribute work to process participants in order to execute a business process from start to end; (iii) the ability to automatically retrieve and store data required for the execution of the process and to delegate (automated) activities to software applications across the organization. Altogether, the engine is continuously monitoring the progress of different cases and coordinating which activities to work on next by generating *work items*, i.e. instances of process activities that need to be taken care of for

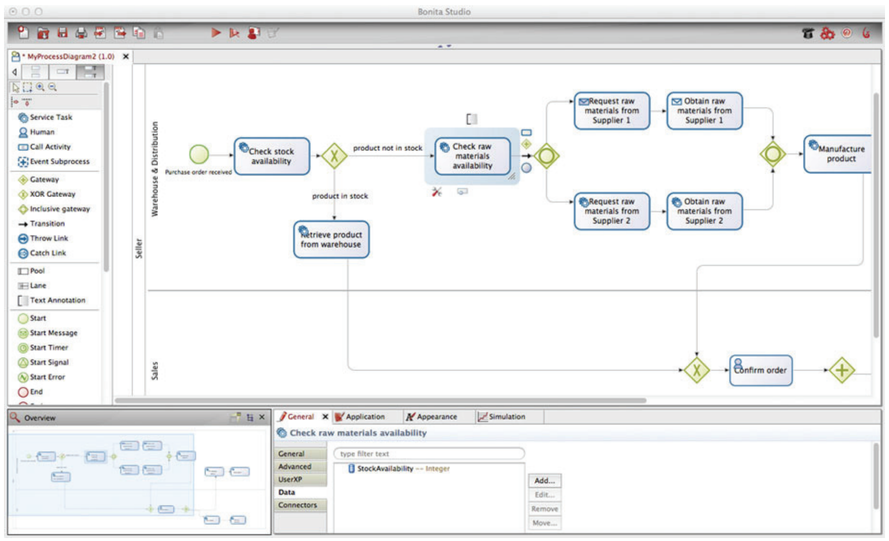
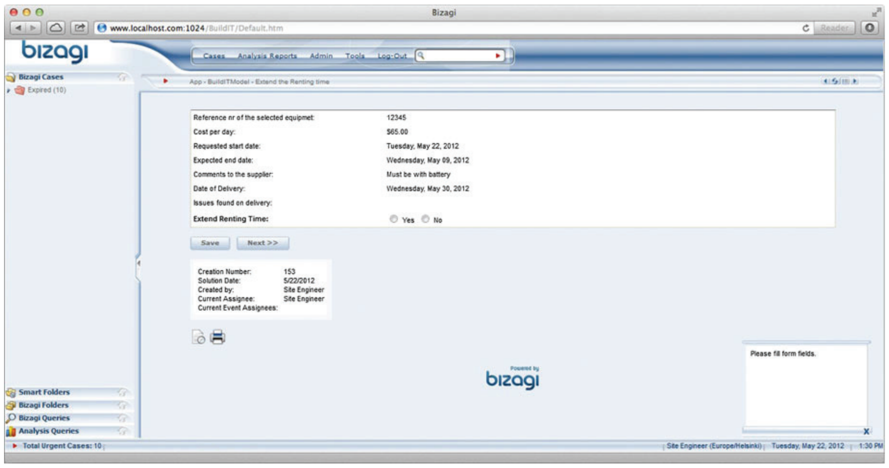


Fig. 9.2 The process modeling tool of Bonita Open Solution from Bonita Soft

specific cases. Work items are then allocated to resources which are both qualified and authorized to carry out these. More specifically, the execution engine interacts with the other components, as discussed next.

Process modeling tool The process modeling tool component offers functionality such as (i) the ability for users to create and modify process models; (ii) the ability to annotate process models with additional data, such as data input and output, participants, business rules associated with activities, or performance measures associated with a process or an activity; and (iii) the ability to store, share and retrieve process models from a *process model repository*. A process model can be *deployed* to the engine in order to be executed. This can either be done directly from the modeling tool or from the repository. The engine uses the process model to determine the temporal and logical order in which the activities of a process have to be executed. On that basis, it determines which work items should be generated and to whom they should be allocated or which external services should be called. Figure 9.2 shows the process modeling tool of Bonita Open Solution from Bonita Soft.

Worklist handler A worklist handler is the component of a BPMS through which process participants are (i) offered work items and (ii) commit to these. It is the execution engine that keeps track of which work items are due and makes them available through the worklist handlers of individual process participants. The standard worklist handler of a BPMS can best be imagined as an *inbox*, similar to that of an email client. Through an inbox, participants can see what work items are ready for them to be executed. The worklist handler might use electronic forms for an activity's input and output data. When a work item of this activity is selected and started by the participant from their worklist, the corre-



Reference nr of the selected equipmet:	12345
Cost per day:	$65.00
Requested start date:	Tuesday, May 22, 2012
Expected end date:	Wednesday, May 09, 2012
Comments to the supplier:	Must be with battery
Date of Delivery:	Wednesday, May 30, 2012
Issues found on delivery:	
Extend Renting Time:	Yes No

Creation Number:	153
Solution Date:	5/22/2012
Created by:	Site Engineer
Current Assignee:	Site Engineer
Current Event Assignees:	

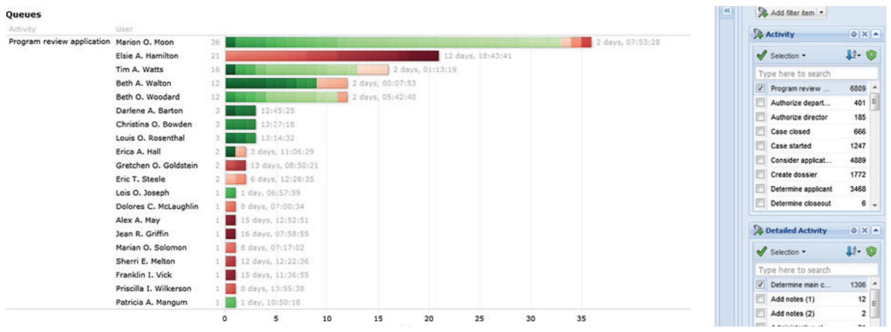
Please fill form fields.

Total Urgent Cases: 10 | Site Engineer (Europe/Helsinki) | Tuesday, May 22, 2012 | 1:30 PM

Fig. 9.3 The worklist handler of Bizagi's BPM Suite

sponding electronic form is rendered on screen. This step is called *check-out*. Participants can then enter data into the form, and signal completion to the engine. This step is called *check-in*. Afterwards, the engine determines the next work items that must be performed for the case in question. Often, participants can to some extent exert control over the work items in their worklist, e.g. with respect to the order in which they are displayed and the priority they assign to these work items. Also, a worklist handler will typically support a process participant in temporarily suspending work items or passing on control to someone else. Which exact features are available depends on the BPMS in question and its specific configuration. It is fairly common to customize worklist handlers, for example according to corporate design, to foster its efficient usage and acceptance within an organization. Figure 9.3 shows the worklist handler of Bizagi's BPM Suite.

External services It may be useful to involve external applications in the execution of a business process. In many business processes, there are activities which are not to be executed in a completely manual fashion. Some of these activities can be performed fully automatically, such that the execution engine can simply call an external application, for example to assess the creditworthiness of a client. The external application has to expose a service interface with which the engine can interact. Thus, we simply refer to such applications as *external services*. The execution engine provides the invoked service with the necessary data it will need for performing the activity for a specific case. On completion of the request, the service will return the outcome to the engine and signal that the work item is completed. This, again, is stored in the execution log. Some other activities in a business process are neither completely manual nor completely automated. Instead, such activities are to be performed by process participants with some form of automated support. For this category of activities, the execution engine will invoke the appropriate services with the right parameters, exactly at the moment



Queues

Activity	User		
Program review application	Marion O. Moon	36	2 days, 07:53:28
	Elsie A. Hamilton	21	12 days, 18:43:41
	Tim A. Watts	16	2 days, 01:13:19
	Beth A. Walton	12	2 days, 00:07:53
	Beth O. Woodard	12	2 days, 05:42:40
	Darlene A. Barton	3	12:45:25
	Christina O. Bowden	3	13:37:18
	Louis O. Rosenthal	3	13:14:32
	Erica A. Hall	2	3 days, 11:06:29
	Gretchen O. Goldstein	2	13 days, 08:50:21
	Eric T. Steele	2	6 days, 12:26:35
	Lois O. Joseph	1	1 day, 06:57:59
	Dolores C. McLaughlin	1	8 days, 07:00:34
	Alex A. May	1	15 days, 12:52:51
	Jean R. Griffin	1	16 days, 07:58:55
	Marian O. Solomon	1	8 days, 07:17:02
	Sherri E. Melton	1	12 days, 12:22:36
	Franklin I. Vick	1	15 days, 11:36:55
	Priscilla I. Wilkerson	1	8 days, 13:55:38
	Patricia A. Mangum	1	1 day, 10:50:18

✓	Program review ...	6809
	Authorize depart...	401
	Authorize director	185
	Case closed	666
	Case started	1247
	Consider applicat...	4889
	Create dossier	1772
	Determine applicant	3468
	Determine closeout	6

✓	Determine main c...	1306
	Add notes (1)	12
	Add notes (2)	2

Fig. 9.4 The monitoring tool of Perceptive Software's BPMOne

that the employee selects a particular work item to work on. A typical example would be the invocation of a Document Management System (DMS) to display to the process participant a file that is important to carry out a specific work item. Think of the equipment rental request, where this would help a clerk to determine the appropriate piece of equipment to order. Sometimes too, a BPMS may need to transfer control over cases between different organizational units or organizations. One way of achieving this is by interacting with an external BPMS which exposes a service interface for this purpose. For example, consider a global insurance company that has offices in three different time zones: Japan, the UK and California. At the end of the working day in each of these time zones, all work items can be transferred to the execution engine in the next zone where the work day has just started. In this way, the execution of the business process never stops.

Administration and monitoring tools Administration and monitoring tools are the tools necessary for the administration of all operational matters of a BPMS. Consider, as a prime example, the actual availability of specific participants. If someone is unavailable to work because of illness or a vacation, the BPMS has to be made aware of this fact in order to avoid allocating work items to such a person. Administration tools are also required to deal with exceptional situations, for example to remove outdated work items from the system. Administration tools are also equipped with process monitoring functionality. One can use these tools to monitor the performance of the running business processes, in particular with respect to the progress of individual cases. These tools can aggregate data from different cases, such as average cycle times of cases, or the fraction of cases that are delivered too late. The BPMS records the execution of a process model step by step. The execution-related events recorded in this way are stored and can be exported in the form of *execution logs*. Some monitoring tools can analyze historical data extracted from the logs and compare it with live data (cf. Chap. 10). Figure 9.4 shows the monitoring tool of Perceptive Software's BPMOne.

This generic architecture, which underpins the functioning of any BPMS, is the evolution of a reference model for WfMSs which was proposed by the Workflow

Management Coalition (WfMC) in the nineties. A dedicated box "WfMC Reference Model" expands on this model.

To illustrate how a BPMS works, recall BuildIT's business process for renting equipment from Chap. 1. Let us suppose it is supported by a BPMS. The execution engine can track that for orders # 1,220 and #1,230 *site engineers* have already filled out the equipment rental requests. On the basis of a process model of the renting equipment process, the execution engine can detect that for both of these cases the proper piece of equipment must be determined. This needs to be done by any of the clerks at the depot. Therefore, the BPMS passes on the request to all worklist handlers of all clerks for further processing. For order #1,240 on the other hand, the equipment rental request is not available yet. So, the BPMS engine will not pass on a similar request for this order yet. Instead, it will await the completion of this work item.

Exercise 9.1 In which state is the process after all the actions of the rental process of BuildIT have been performed as described above? Which work items can you identify that are under control of the BPMS? Make sure to identify both the case and the activity for each work item.

WfMC REFERENCE MODEL

The Workflow Management Coalition (WfMC) is a standardization organization, founded in 1993, in which BPMS vendors, users and researchers have a seat. The purpose of the WfMC is to achieve generally accepted standards for terminology and interfaces for the components of a BPMS [35].

The WfMC has produced the so-called *WfMC reference model* which has become well-established in the world of process automation. The idea behind this reference model is that any supplier of a BPMS can explain the functioning of its specific system on the basis of it. The original reference model included six components, which resemble the components of the BPMS architecture in Fig. 9.1. They are: workflow engine, process modeling tools, administration and monitoring tools, worklist handler, external applications and external BPMSs. In the reference model, the interactions between its components take place through so-called *interfaces*, which are numbered from 1 to 5. Three of these interfaces are still recognizable in the BPMS architecture that is discussed in this chapter:

- *Interface 1* concerns the interaction between the engine and process modeling tools,
- *Interface 2* concerns the interaction between the engine and the worklist handler,
- *Interface 5* concerns the interaction between the engine and the administration and monitoring tools.

The other interfaces of the WfMC reference model are subsumed by recent developments. *Interface 3* governed the interaction between IT applications and the execution engine, but this has become outdated by the advent of standard service interfaces over the Web (e.g. Web services). Similarly, *interface 4*—which addressed the integration with external BPMSs—has also been subsumed since this can also be realized through standard service interfaces.

While most components of the WfMC reference model reappear in the BPMS architecture of Fig. 9.1, it should be noted that the WfMC architecture does not include the process model repository and does not explicitly represent execution logs. These elements, however, have become crucial assets in the area of process automation, analysis and redesign.

Exercise 9.2 Consider the following questions about a BPMS:

- Why would it not be sufficient to only create a business process model with the modeling tools, without any information on the types of resource that are available?
- In what situation will the execution engine generate multiple work items on the basis of the completion of a single work item?
- Can you provide examples of external services that may be useful to be invoked when a participant wishes to carry out a work item?
- What would be a minimal requirement to be able to pass on work items between the BPMSs of the insurance company that was provided as an example?
- If it is important that a BPMS hands out work items to available resources, can you imagine other, relevant types of information on resources that are useful to be captured by an administration tool (apart from whether they are ill or on vacation)?

9.1.3 The Case of ACNS

Building on the explanation of the BPMS architecture in the previous section it is now possible to sketch an example of an operational BPMS. We use a simplified view on a process in which claims are assessed within the ACNS company (*A Claim is No Shame*). The first activity in this process is an assessment of the claim, which is to be done by a senior acceptor or a regular acceptor. A regular acceptor is only qualified to make this assessment in the situation where the claim amount of a case is below €1,000. In case of a negative assessment, it is the responsibility of the account manager to convey the bad news to the customer. In case of a positive assessment, an electronic invoice is to be generated by a clerk of the finance department, who needs to dispatch that to the client in question. After these activities, the process is completed.

The above description shows that there are two dimensions that must be covered with the process modeling tool of the BPMS: (1) the procedure that specifies the various activities and (2) the various participants who are involved in carrying out these activities. The former part is recorded in a process model; the second is captured in, what is often referred to as, a *resource classification*. In addition, the relations between these models or specifications must be defined, i.e. who is able and qualified to perform which activity. Often, these relations are also specified as part of the process model. These relationships may not be static but be dependent on all kinds of business rule. For example, the distinction between the authorization levels of the senior and ordinary acceptor in assessing claims is an example of a dynamic rule, i.e. it is determined by the current value one of a piece of information.

Once these descriptions are defined, the execution engine of a BPMS would generally be able to support this process. Now let us assume that almost simultaneously two claims come in:

1. A car damage of €12,500, as claimed by Mr. Bouman.
2. A car damage of €500, as claimed by Mrs. Fillers.

Ms. Withagen has been with ACNS for a long time and, for the past years, functions on the level of a senior acceptor. This month, Mr. Trienekens has started his training and works as an acceptor. At the start of his contract, the system administrator, Mr. Verbeek, has used the administration tool of the BPMS to add Mr. Trienekens to the pool of available acceptors.

Based on the process model, the resource classification, and operational data on the availability of the various employees, the enactment service of the BPMS now takes care of forwarding both newly received claims to the worklist handlers of Ms. Withagen. After all, she may assess both claims based on her qualifications. Mr. Trienekens, in his turn, will only see in his worklist handler the damage claim of Mrs. Fillers.

On noting the work item in his worklist, Mr. Trienekens starts to work on it immediately. He selects the damage claim of Ms. Fillers to deal with. In response to that action, the execution engine ensures that the corresponding work item *disappears* from the worklist of Ms. Withagen. The reason, of course, is that this piece of work needs to be carried out only once. In any case, Ms. Withagen is still working on the handling of an earlier case, but shortly thereafter selects the claim of Mr. Bouman to deal with through her worklist handler.

In response to the selection of work items by both Mr. Trienekens and Mrs. Withagen, the execution engine will ensure that both will see the electronic claim file on their screen of the respective customers. The execution engine does so by using the appropriate parameters in invoking the DMS of ACNS at the workstations of the acceptors. The DMS also displays the scanned version of the claims, which were originally sent in on paper. In addition, the BPMS takes care of displaying to both the acceptors an electronic form that they can use to record their assessment, also through the invocation of a service.

Mr. Trienekens decides to reject the claim. The worklist handler notices this, because it monitors the specific field on the electronic form that receives a negative value. Based on the logic captured in the process model, the execution engine can determine that the case must be handed over to the account manager of Ms. Fillers and sends a work item to that participant, requesting to inform the client on the negative assessment.

Ms. Withagen arrives at a positive assessment of the claim under her watch and decides to approve it. The execution engine ensures that a service is invoked to determine the new monthly premium for Mr. Bouman, taking into account his no-claim history which is registered in a claim database. The retrieval of the information from this database is also realized through a service. Once this calculation is completed, a work item for the various available financial employees is created to pay the damage. The work item appears in the worklist of each of these financial officers, until one of them selects it for processing. After the payment has been carried out, the process is completed.

As can be seen in the ANCS example, all components of the BPMS architecture play a role in coordinating the work, specifically to ensure that the appropriate work items are created and carried out by the involved participants.

Exercise 9.3 Consider the following developments and indicate which components of the BPMS architecture are affected when they are to be taken into account:

1. A new decision support system is developed to support acceptors in making their assessment of claims.
2. Ms. Withagen retires.
3. A new distinction between claims becomes relevant: regular acceptors are now also qualified to deal with claims above €1,000 as long as they worked on previous claims by the same client.
4. Claims that are issued on cars which are over 10 years old need to be continuously monitored by management.

Up to this point, we have discussed BPMSs as if they are particular software packages that deliver more or less the same functionality. However, it is closer to the truth to state that there are distinct flavors of BPMSs and that different organizations or different processes within the same organization may be best served by different types of BPMS. This observation is further discussed in the box "Types of BPMS".

TYPES OF BPMS
There are several ways to distinguish between the available BPMSs. One classification is based on the use of two axes: one that captures the *degree of support* that the BPMS delivers and the other that expresses how that systems differ from each other with respect to their *orientation on process or data*. We describe and illustrate four different types of system: groupware systems, adhoc workflow systems, production workflow systems, and case handling systems. These systems can be positioned in the spectrum of BPMSs as shown in Fig. 9.5.

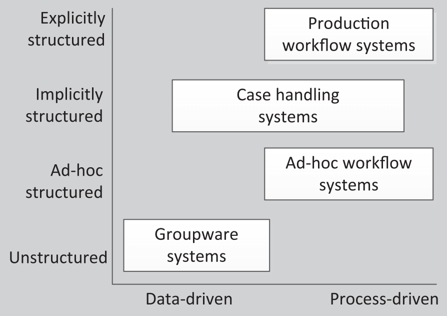


Fig. 9.5 The spectrum of BPMS types

Groupware systems: The two underlying principles of groupware systems are that the user is enabled to easily share documents and information on the one hand and to directly communicate with other users on the other. The best known example of a groupware system is IBM's Lotus Notes. Groupware systems are very widely used and particularly popular for their high operational flexibility. When it comes to supporting business processes, these systems cannot do such in a strict sense. Groupware systems traditionally do not support an explicit notion of the business processes; however, extensions like Lotus Domino Workflow exist as well.

Ad-hoc workflow systems: Ad-hoc BPMSs, like TIBCO's BusinessWorks or Comalatech's Ad-hoc Workflows, allow on-the-fly process definitions that can be created and modified. So, even though there is already a process definition in existence to deal with a case type, it is possible during the execution of that process to adapt the process, for example to include an additional step. For these kinds of system, it is often so that each case (an order, claim, complaint, etc.) has its own private process definition. This prevents many problems that are acclimated with updating a general process definition when it has running instances. Clearly, ad-hoc BPMSs allow much flexibility. The BPMS InConcert, for example, even allows users to trigger a business process instance on the basis of a completely empty process definition, which is extended as it becomes clearer what needs to happen and in what order. Another direction is that initially the BPMS works on the basis of a standard solution or template, which can be modified at will. Interestingly, such a modified procedure may be used as the template for starting the processing of a new case. In general, there are two major requirements to successfully apply ad-hoc BPMSs in an organization. The first requirement is that end users are very aware of the processes in which they operate. This means that only processes should be defined or modified by people with a very good overview of the process and the consequences of deviating from usual practice. The second requirement is that users have sophisticated tools at their disposal to model business processes and that they are capable of modeling. The combination of these requirements hinders the wide-scale application of ad-hoc BPMSs at this point.

Production workflow systems: The most familiar type of BPMS is the production workflow system. Typical representatives are IBM's Business Process Manager and Bizagi's BPM Suite. Much of what we described in the previous sections on workflow applies to this class of BPMSs. Work is routed strictly on the basis of explicitly defined process descriptions captured in process models. Managing data, such as documents or e-mail support, is generally not offered by these systems. In general, it is also impossible or hard to deviate from a process logic if that has not been explicitly captured in the process model. Sometimes a further distinction is made within production workflow into *administrative* and *transaction processing* BPMSs. This makes it possible to distinguish further shades with respect to the degree of automation of the work that is coordinated. Administrative BPMSs, are used in settings where a (large) portion of work is still performed by people; transaction processing BPMSs support business processes that are (almost) fully automated.

Case handling systems: The idea behind a case handling system (or Adaptive Case Management system) is that it does not strive for a tight and complete specification of a business process in a model. Rather, *implicit* process models are used, which capture a conventional flow from which—unless this is explicitly prohibited—a user can deviate. A case handling system is usually fully aware of the precise details of the data belonging to a case (including customer data, financial or medical data). On the basis of such awareness, the system is able to provide end users a highly precise insight into the status of a case, as well as the most obvious steps to continue the process. Contemporary examples are i-Sight's Case Management Software and BPMOne by Perceptive Software. The latter also, if desired, supports a production workflow approach and in that sense is a hybrid BPMS.

There are other types of system that share characteristics and functionality with BPMSs. *Document management systems* (DMSs) primarily take care of the storage and retrieval of documents, like document scans and PDFs, but they often offer workflow automation features as well. An example is Adobe's LiveCycle Enterprise Suite. *Process Orchestration Servers* focus on process automation but have a specific emphasis on automated processes that require the integration of multiple enterprise applications. An example is Oracle's SOA Suite.

9.2 Advantages of Introducing a BPMS

In this section we reflect on why it would be attractive for organizations to use a BPMS. There are four broad categories of advantages which we will discuss here: workload reduction, flexible system integration, execution transparency, and rule enforcement.

9.2.1 Workload Reduction

A BPMS automates part of the work that is done by people in settings where such a system is not in place. First of all, it will take care of *transporting work* itself. In a paper-based organization, work is usually transported by internal postal services, often delaying processing for one work day at each handover, or by the participants themselves at the expense of their working time. All such delays are completely eradicated when a BPMS can be used to dispatch work items electronically. In some situations, the BPMS can take care of the entire process by invoking fully automated applications. In such cases, we speak of *Straight-Through-Processing* (STP). Particularly in the financial services, many business processes that used to involve human operations are now in STP mode and coordinated by BPMSs. Also in other domains, think for instance of electronic visa, at least a portion of the cases can be handled in a completely automatic fashion.

The second type of work that is being taken over by the BPMS concerns *coordination*. The BPMS uses the process model for determining which activities need to be performed and in what order. So, every time the BPMS uses this knowledge to route a work item it potentially saves someone the time to even think about what should be done next. Another form of coordination time saved is the signaling of completed work. In a paper-based organization, work will be lying around for quite some time in case of work hand-overs. What often happens is that someone takes over work, suspends it for some reason, after which the work package gets stuck in another pile of work. The BPMS will at all times be able to signal the status of all work items and it can take actions to ensure that progress is being made.

The final type of workload reduction by using a BPMS is the *gathering of all relevant information* to carry out a particular task. In a situation without a BPMS, it is the employee who needs to do this collection. Finding the right file in particular—it is never there where you would expect it—can be a time-consuming affair. Note that this type of advantage rests on the assumption that along with the introduction of the BPMS the effort is taken to digitalize the stream of documents in an organization. The implementation of a DMS is actually what is often observed alongside a BPMS implementation. Certain vendors, such as IBM and Perceptive Software, offer integrated suites of BPMS and DMS functionality. Other BPMS vendors often have strategic cooperations with companies that offer a DMS, such that it is relatively easy to integrate their joint systems.

9.2.2 Flexible System Integration

Originally, the most mentioned argument to start with a BPMS is the increased flexibility that organizations achieve with this technology. To explain this best, a short reflection on the history of computer applications is due. There is an interesting trend, as identified by Van der Aalst and Van Hee [95] that generic functionality is split off from applications at some point.

Roughly throughout the 1965–1975 period, computer applications were run directly on the operating systems (OS) of a computer. Each application would take care of its own data management and would be using proprietary techniques to do this efficiently. As a result, it turned out to be difficult to share data among applications and to maintain consistency. Clearly, programmers of different applications would be involved with developing similar routines to solve similar problems. From 1975 onwards, DBMSs as a new type of standard software emerged that took on the generic task of managing data efficiently. As a result, data could be shared rather easily and programmers of new applications would not need to worry anymore about ways to store, query, or retrieve date. Some 10 years later, around 1985, User Interface Management Systems (UIMS) were introduced to provide a very generic interface component to many applications. Through the provision of facilities like drop-down boxes or radio buttons in accessible libraries, each computer programmer would be able to make use of these. By 1995, the first commercial BPMSs enter the market place (considerable time before, research prototypes have been available). Like DBMSs and UIMSs in their focus area, BPMSs would provide generic support for the area of business process logic.

The introduction of a BPMS is a logical sequel to the separation of generic functionality of what were one monolithic computer programs. Still in the 1990s, it was estimated that 40 % of all the lines of code running on the mainframe computers of banks would have to do with business process logic, not with the calculations or data processing themselves. The typical kind of information processing in the context relates to the identification of activities, their order of execution, and or the participants responsible for carrying them out. For example, it would be specified that after a mortgage offering was completed, this needed to be signaled to the manager of the department, triggering a signal on her monitor.

The obvious advantage related to this development is that it has become much easier with a BPMS to manage business process logic on its own. This is due to the fact that it is much more convenient to update the description of a business process without having to inspect the application code. Also, the reverse would become easier, i.e. modifying an application while not touching on the order of how things on the business process level would need to unfold. BPMSs, in short, would enable organizations to become more flexible in managing and updating their business processes as well as their applications.

BPMSs also provide the means to glue together separate systems. Large service organizations typically deploy myriads of IT systems, which all exist more or less independent of each other. Often, such a situation is referred to as *island automation*. A BPMS may be introduced in such a situation as a means of integrating such systems. It will safeguard that all the separate systems will play their due role in the business process they support.

A word of caution is due here. The BPMS itself will offer no direct solution to the problem that there is often a redundant storage of information across many different IT systems. In fact, a BPMS will in general have no knowledge of the actual data that end users will manipulate using the various IT systems. If the BPMS is to operate as an integrator between all the existing systems, this will require a thorough information analysis to map which data is used and available.

9.2.3 Execution Transparency

An advantage that is often overlooked is that a BPMS can operate as a treasure trove with respect to the way that business processes are really executed. It is likely that the developers of the first BPMSs did not clearly have any advantage in mind on providing any insights about the execution of a process to anyone else than people executing the process itself. Sure enough, to have a BPMS operate at all, it must keep an accurate administration on which work items are due, which can only be determined by actively monitoring and recording which work items have been completed by which resources and at what time new cases enter the process. Yet, for a BPMS to function properly, it is not necessary to keep all that data available once the associated cases are completed. The management overseeing such a process, however, may have an entirely different perspective on this issue. There are two types of information that may be useful to generate interesting insights on basis of the administration a BPMS keeps:

1. *Operational information*, which relates to recent, running cases, and
2. *Historic information*, which relates to completed cases

Operational information is relevant for the management of individual cases, participants, or specific parts of a business process. A characteristic example is the following. From analyses of various governmental agencies involved with granting permits in the Netherlands it has become clear that determining the exact status of a permit application was one of the most time-consuming activities for the civil servants involved. By the use of most commercially available BPMSs, retrieving that status is a futility. Such a status may be, for example, that the request of Mr. Benders to extend his house with an extra garden wing has been received, matched with the development plans for the area he is living in, and that further processing is now dependent on the receipt of the advice of an external expert. Another use of operational information would relate to the length of queue in terms of work items. For example, there are 29 applications for building permits that await the advice of an external expert. From these examples it may become clear that initiatives to improve customer service, in particular with respect to answering questions about their orders, often relies on the use of a BPMS.

Historic information, in contrast to operation information, is often of interest on a particular level of aggregation, for example covering more cases over an extended period of time. This kind of information is of the utmost importance to determine the performance of a particular process or its conformance to particular rules. With respect to the former, you may want to think of average cycle times, the number of completed cases over a particular period, or the utilization of resources. The latter category could cover issues like the kind of exceptions that have been generated or the number of cases that violated a particular deadline.

It makes sense to consider the kind of insights that need to be retrieved from a BPMS before it is actually implemented in an organization. Technical issues play a role here, like the period of time that the logs of a BPMS need to be kept and, therefore, how much storage space should be devoted for that. Consider that it becomes a bit problematic if historic information is important on the aggregate level of years if there is only space to save the events of at most a months. There are also conceptual issues. If it is important to monitor a certain milestone within a process it is essential that it makes part of the process model that is used for the execution of the related business process. To use the previous example: If it is important to be able to recognize the stage in which a case has to wait for the advice of an external expert, then that milestone must be part of the process model. In this way, process automation provides the foundation for process intelligence (cf. Chap. 10).

9.2.4 Rule Enforcement

Except for the obvious advantage that a business process could be executed more efficiently by using a BPMS, such a system will also take care of that the process is carried out in precisely the way that it has been designed. When rules are explicitly enforced, this can be considered as a quality benefit: one does, what one promises. In many settings, employees will have considerable freedom to carry out a business process in the way it looks best to them (or most convenient). This individual assessment does not necessarily coincide with the best possible way a business process is executed from the overall perspective of an organization. In this respect, a BPMS can be a means to safeguard that business processes are executed in a predefined way, without any concessions.

As an illustration, consider the *separation of duties* control that is well known in the financial services domain. It means that the registration and inspection of a financial transaction will need to be carried out by different individuals. This type of logic is both quite easily implemented in BPMSs and enforced by such systems. The BPMS registers which individuals have carried out which work items and can take this information into account when allocating new work items. Note that a BPMS is, in general, sufficiently sophisticated so that employees can alternatively fulfill the registering and inspecting role for different cases.

The capacity of a BPMS to enforce rules is currently of much interest to organizations. Until a decade ago, primarily governmental organizations were implementing such systems purely motivated by this concern. After all, there are various laws that such organizations have to conform with. Nowadays, financial and other professional service organizations have become similarly enamored by BPMSs. An important development in this is the rise of various governance frameworks, which started in 2002 with the *Sarbanes–Oxley Act* as a reaction to misconduct in Enron and Worldcom. The law places a high responsibility with company executives to install management controls and procedures within organizations and to see their proper execution. Obviously, this is were BPMSs can play an important role.

Exercise 9.4 To which categories would you classify the following incentives to introduce a BPMS in an organization?

- An auditing agency has found that the written procedures and actual execution of business processes are not aligned. The management of that organization wishes to enforce the written procedures and decides to introduce a BPMS.
- The clients of an organization complain that they can only get very shallow updates on the progress of the orders they make. The IT manager of that organization looks into the use of a BPMS to capture and provide status information of all these order.
- An insurance organization finds out that there is a high need to quickly adjust the way claims processing is carried out to the offerings that its competitors bring to the market. Using a BPMS is considered to address this demand.

9.3 Challenges of Introducing a BPMS

Despite the many advantages of using BPMSs, there are some notable obstacles with respect to introducing these in an organization. We will distinguish between technical and organizational challenges here.

9.3.1 Technical Challenges

What should be one of the strengths of a BPMS is also one of the pitfalls. A BPMS is capable of integrating the use of different types of information system in their support of a business process. The challenge is that many applications have never been developed with this coordinated use in mind. The mainframe applications that can still be found within banks and insurance companies today are notorious in this regard. In the most favorable case, such systems are technically documented but it happens often that there is no one of the original development team available anymore who knows exactly how these are structured. In such cases, it is very hard to determine how a BPMS can trigger such systems to make them support the execution of a particular work item, to exchange information between the BPMS and such a system (e.g. case data), and how to determine when an employee has used such a system to complete a particular work item.

A technique that has been used to make interaction with such legacy systems at all possible is *screen scraping*. The interaction between a BPMS and the mainframe application then takes place on the level of the user interface: the key strokes that an end user should make are emulated by the BPMS and the signals sent to the display are tracked to establish the progress of carrying out an activity. It will come as no surprise that such low-level integration solutions will incur much rigidity to the overall solution and will, in fact, undermine the flexibility advantages that are normally associated with using a BPMS.

A specific problem that occurs with respect to the integration of existing applications with BPMSs is the lack of process-awareness of traditional systems. In a process-aware system, separate cases will be handled separately. In other words, such a system works on a case-by-case basis. In many traditional systems, *batch processing* is the dominant paradigm. This means that a particular task is executed for a potentially large set of cases, which does not always go well with the philosophy of a BPMS. Note how the Case-based work heuristic mentioned in Chap. 8 explicitly targets this situation.

Fortunately, in the area of system integration much progress has been made in the past decade. Many old systems are being phased out and new, open systems with clearly defined interfaces take their places. Technologies that are referred to as *Middleware* and *Enterprise Application Integration* tools are now available that strongly facilitate the communication and management of data in distributed applications. Microsoft's BizTalk and IBM's WebSphere are well-known software suites that can be used in this respect and there are open source technologies available as well. The success of *Web services* is another driver behind improved, coordinated use of different types of information system, including BPMSs. A Web service is a piece of functionality, for example the identification of the best possible price for a particular good within a range of providers of that good, which can be invoked over the Internet. Most BPMSs provide good support for integrating specific Web services in executable business processes. This kind of set-up would fit within a popular software architecture paradigm that is commonly referred to as *Service-Oriented Architecture*.

With respect to technical integration capabilities it is fair to say that recent developments are favorable for the use of BPMSs and that technical challenges, at least with respect to this aspect, are likely to further decrease over the next years.

9.3.2 Organizational Challenges

The introduction of an operational BPMS often has an impact on extensive parts of an organization. This implies that the introduction of a BPMS can be challenging from an organizational perspective. The interests of different stakeholders have to be balanced, who usually have diverging performance objectives and vie for the same resources. Getting an insight into how existing processes unfold is an enormous challenge in itself, sometimes taking months of work. Here, not only political motives may play a role—not everyone will be happy to give away how work is done, especially if not much work is done at all—but psychological ones as well: people tend to focus on describing the worst possible exceptions when asked to describe what their role is in a process. One scholar has referred to this tendency as the reason "why modelers wreck workflow innovation".

A factor that adds to this complexity is that organizations are dynamic entities. It is fairly usual that during the introduction of a BPMS, which may span a couple of months, organizational rules change, departments are scrapped or combined, participants get other responsibilities, new products are introduced or taken off the market. These are all examples of events that may be important to consider when the aim is to make a BPMS function properly in an organizational setting. In practice, this accounts for the insight that the gradual introduction of a BPMS is usually more successful than a "big bang" strategy, in which a BPMS from one day on the other is expected to replace the way operations were managed.

The perspective of the users on the introduction of a BPMS should be considered carefully. Most subject experts will first need to experience *hands-on* what it is to use a BPMS before they can really appreciate what that means for their job. There may also be concerns and fears. First, there might be a "Big brother is watching you" sentiment. Indeed, a BPMS will record all the events that are involved with executing a process, including who carried out what piece of work and at what time. It makes no use—and from a change management perspective, it could actually be self-defeating—to ignore this concern. Rather it is up to organizations to clarify how this information will be used and that there are positive effects that can be expected of the usage of this information as well. Another fear that is common with end users of BPMSs is that their work will take on a mechanistic trait, almost as if they are working on a chain gang. This fear is in part genuine. It is true that the BPMS will take care of the allocation and routing of work. What can be argued, though, is that these are not the most exciting or valuable parts of the work that needs to be done (which is precisely the reason that they could be automated in the first place). If you would consider the situation where an employee needs to spend large parts of their time on finding the right information to do the job properly, the BPMS can be a favorable mechanism to give that time back to the employee. Another line of reasoning is that it highly depends on the configuration of the BPMS whether the mechanization effect will actually occur. Compare, for example, the situation where a BPMS pushes a single work item at a time to an employee to be carried out or rather a range of work items that someone could choose according to one's own preferences. These options, which result from a configuration decision, can make a huge difference in the perception of the value of the BPMS.

To sum up, the introduction of a BPMS is particularly complex, precisely because it supports entire business processes. It is not for nothing that out of many research projects into IT projects "strong management commitment" is always on top of the factors that explain successful implementations. The introduction of a BPMS is, perhaps even more so than for other types of technology, not for the faint of heart.

Exercise 9.5 Consider the following issues that come up when introducing a BPMS in a hospital to support preoperative care, i.e. the preparation and management of a patient prior to surgery. Classify these to the technical or organizational issues, or both.

1. On hearing about the plans to introduce a BPMS, the surgeons flatly reject to cooperate on this endeavor. Their claim is that each patient is an individual person that cannot be trusted to the care of a one-size-fits-all system.
2. The anesthetists in the hospital use a decision support system that monitors the proper dosage of anesthetics to patients. The system is developed as a stand-alone system that is difficult to synchronized with the BPMS, which has to feed the decision support system with patient data.
3. The nurses are provided with mobile devices, which they can use to access their worklist handlers. However, they find it difficult to follow up on the automatic notifications which are signaled to them as gentle vibrations of the device.

9.4 Turning Process Models Executable

In this section we will show how to automate a business-oriented process model in order to execute it on a BPMS. In particular, we will consider the case of production BPMSs.

Mapping processes for automation requires an approach to process modeling that is quite different from what is needed for communication or analytically focused process models. In fact, because of their intent, business-oriented process models are not necessarily precise and may thus contain ambiguities. Conversely, *executable process models* must be precise specifications in order to be interpreted by a BPMS.

We propose a five-step method to incrementally transform a business-oriented process model into an executable one, as follows:

1. Identify the automation boundaries
2. Review manual tasks
3. Complete the process model
4. Bring the process model to an adequate granularity level
5. Specify execution properties

Through these steps, the business-oriented model will become less abstract and more IT-oriented. These steps should be carried out on a process model that is both correct in terms of structure and behavior. For example, if the model contains behavioral errors like a deadlock, the BPMS may get stuck while executing an instance of this process model, with a potential impact on the operations of the organization. We have already discussed process model verification in Sect. 5.4. From now on we assume that the process model is correct.

9.4.1 Identify the Automation Boundaries

First, we need to identify what parts of our process can be coordinated by the BPMS, and what parts cannot. In a process there are *automated*, *manual* and *user* tasks. Automated tasks are performed by the BPMS itself or by an external service while manual tasks are performed by process participants without the aid of any software. A user task sits in-between an automated and a manual task. It is a task performed by a participant with the assistance of the worklist handler of the BPMS or of an external task list manager.

This difference between automated, manual and user tasks is relevant: automated and user tasks can easily be coordinated by a BPMS, while manual tasks cannot. Thus, in this first step we need to identify the type of each task. Then, in the next step, we will review the manual tasks and assess whether we can find a way to hook up these tasks to the BPMS. If this is not possible, we will have to consider whether or not it is convenient to automate the rest of the process without these manual tasks.

Let us consider again the order fulfillment process model that we created in Chap. 3, which is shown in Fig. 9.6 for convenience (for the moment, discard the activity markers). Let us assume we get this model from a business analyst and our task is to automate it from the seller viewpoint. This means we need to focus on the process in the Seller pool and discard the rest. The first activity, "Check stock availability", sits in the ERP lane. This means that already at the conceptual level it was identified as an automated task. ERP systems do provide modules to manage inventories which automatically check the stock levels of a product against a warehouse database. This activity is highly repetitive as it is performed for each purchase order received. Performing it manually would be very inefficient and expensive, since it would employ a process participant though it does not require any particular human skill. Similar considerations hold for "Check raw materials availability", which is also an automated task. Another example is activity "Manufacture product". This is performed by an equipment (the manufacturing plant) which exposes its functionality via a service interface. So from the perspective of a BPMS, this is also an automated activity.

Continuing with our example, there are other tasks such as "Request raw materials from Supplier 1(2)" and "Get shipping address" that are devoted to sending and receiving messages. These are also examples of automated tasks. They can be implemented via automatic e-mail exchange or Web service invocation (and BPMSs typically provide these capabilities), despite they are not explicitly modeled inside a system lane. Recall that we are looking at a business-oriented process model, where it may not be relevant to model via lanes all existing systems (in this case an e-mail service or a Web service).

Other tasks like "Retrieve product from warehouse", "Obtain raw materials from Supplier 1(2)" and "Ship product" are manual. For example, "Retrieve product from warehouse" requires a warehouse worker to physically pick up the product from the shelf for shipping. In the presence of a manual task we have two options: (i) we isolate the task and focus on the automation of the process before and after it, or (ii) we find a way for the BPMS to be notified when the manual task has started or

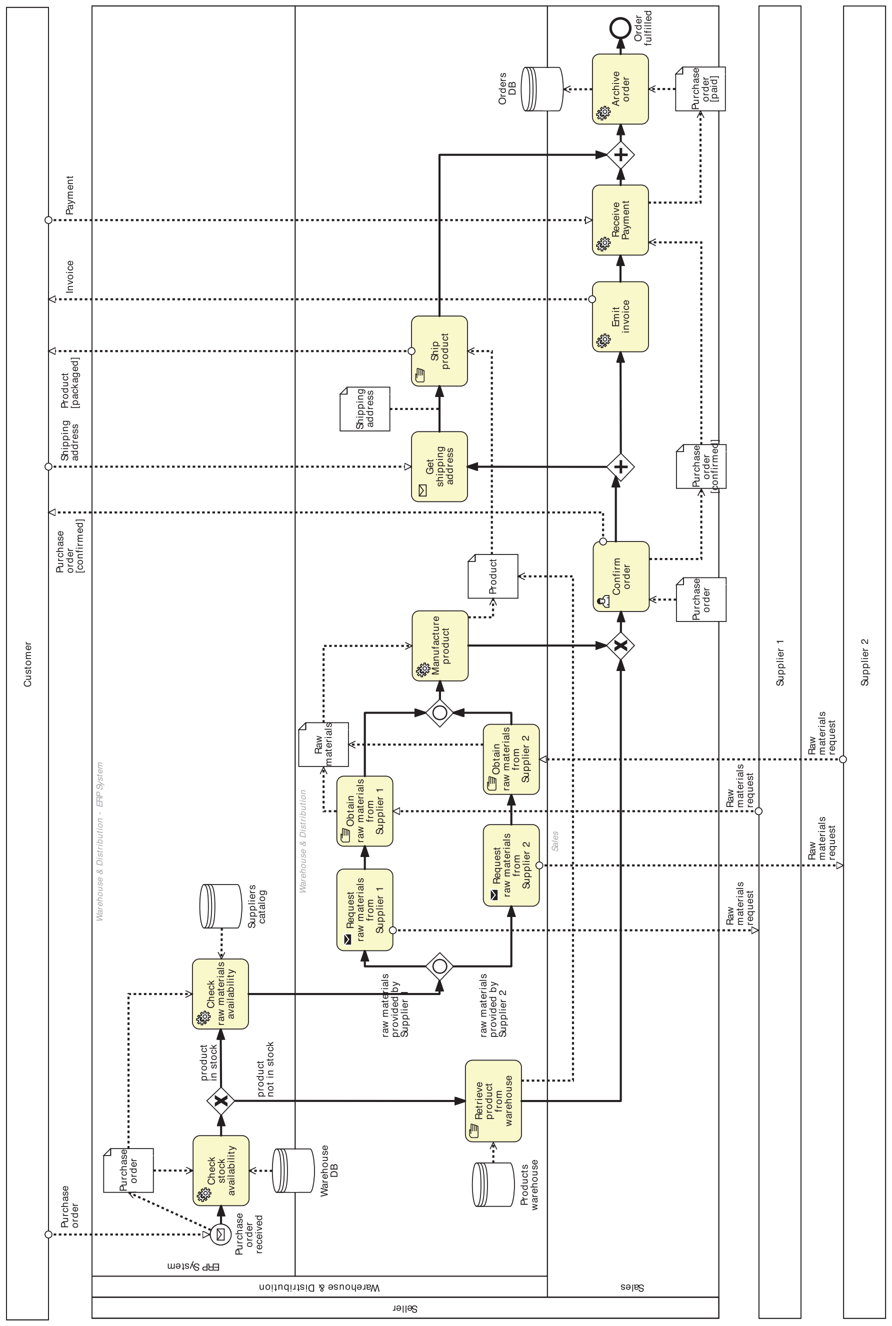


Fig. 9.6 The order fulfillment model that we want to automate

completed. We will get back to this point in the second step. For now, all we need to do is identifying these manual tasks.

“Confirm order” is an example of a user task: it requires somebody in sales (e.g. an order clerk) to verify the purchase order and then confirm that the order is correct. User tasks are typically scheduled through the worklist handler of the BPMS. In our example, an electronic form of the purchase order will be rendered on screen for the order clerk, who will verify that the order is in good state, confirm the order and thus submit the form back to the execution engine.

The distinction between automated, manual and user tasks is captured in BPMN via specific markers on the top-left corner of the task box. Manual tasks are marked with a hand while user tasks are marked with a user icon. Automated tasks are further classified into the following subtypes in BPMN:

- *script* (script marker), if the task executes some code (the script) internally to the BPMS. This task can be used when the functionality is simple and does not require access to an external application, e.g. opening a file or selecting the best quote from a number of suppliers.
- *service* (wheels marker), if the task is executed by an external application, which exposes its functionality via a service interface, e.g. “Check stock availability” in our example.
- *send* (filled envelope marker), if the task sends a message to an external service, e.g. “Request raw materials from Supplier 1”.
- *receive* (empty envelope marker), if the task waits for a message from an external service, e.g. “Get shipping address”.

These markers apply to tasks only. They cannot be used on sub-processes since a sub-process may contain tasks of different types. The relevant markers for our example are shown in Fig. 9.6.

Exercise 9.6 Assume you have to automate the loan assessment process model of Solution 3.7 for the loan provider. Start by classifying the tasks of this process into manual, automated and user ones and represent them with appropriate task markers.

9.4.2 Review Manual Tasks

Once we have identified the type of each task, we need to check whether we can link the manual tasks to the BPMS, so that we can maximize the value obtained by the BPMS. Alternatively, we need to isolate these tasks so that we can automate the rest of our process. There are two ways of linking a manual task to a BPMS: either we implement it via a user task or via an automated task.

If the participant involved in the manual task can notify the BPMS of the task completion using the worklist handler of the BPMS, the manual task can be turned into a user task. For example, the warehouse worker performing task “Retrieve product from warehouse” could check-out a work item of this task from their worklist to indicate that they are about to perform the job, manually retrieve the product from the shelf, and then check-in the work item back into the BPMS engine. Alternatively, check-out and check-in can be combined in a single step whereby the worker simply notifies the worklist handler that the job has been completed.

In some cases, a participant may use the aid of technology integrated with the BPMS to notify the engine of a work item completion. For example, the warehouse worker could use a barcode scanner to scan the barcode of the raw materials being obtained. The scanner would be connected to the BPMS so scanning the barcode would automatically signal the completion of "Obtain raw materials from Supplier 1(2)". In this case, the manual task can be implemented as a receive task awaiting the notification from the scanner, or by a user task handled by a worklist handler which in turn is connected to the scanner. If we use a receive task, the BPMS will only be aware of the work item's completion, in which case informing the warehouse worker that a new work item is available would be outside the scope of the BPMS. If we use a user task, the worker will be notified of the new work item by the BPMS, and will use the scanner to signal the work item's completion to the BPMS engine. Similar considerations hold for task "Ship order". Since each manual task of our example can be linked with a BPMS, this process can be automated as a whole.

Exercise 9.7 Consider the loan assessment model that you obtained in Exercise 9.6. Review the manual tasks of this model in order to link them to a BPMS.

There are cases in which it is not convenient to link manual tasks to a BPMS.

Example 9.1 Let us consider the university admission process described in Exercise 1.1, with the improvements discussed in Solution 1.5. The process until the point where the application is batched for the admissions committee (shown in Fig. 9.7a) can be automated. Once all the applications have been batched, the committee will meet and examine all of them at once. However, this part of the process (shown in Fig. 9.7b) is outside the scope of a BPMS. The tasks required for assessing applications cannot be automated because they involve various human participants who interact on an ad-hoc basis, and it would not be convenient to synchronize all these tasks with the BPMS. Eventually, the committee will draw a list of accepted candidates, transfer it to the admissions office, and a clerk at the admissions office will update the various student records, at which time the rest of the process can proceed within the scope of the BPMS (shown in Fig. 9.7c).

In this example we cannot automate the whole process. So we need to isolate activity "Assess application", an ad-hoc activity containing various manual tasks, and automate the process before and after this activity. An option is to split the model into three fragments as shown in Fig. 9.7 and only automate the first and the third fragment. Another option is to keep one model, and simply remove the ad-hoc activity. Some BPMSs are tolerant to the presence of manual tasks and ad-hoc activities in executable models, and will discard them at deployment time (like comments in a programming language). If this is the case, we can keep these elements in. Observe the use of the untyped event to start the third process model fragment in Fig. 9.7. In

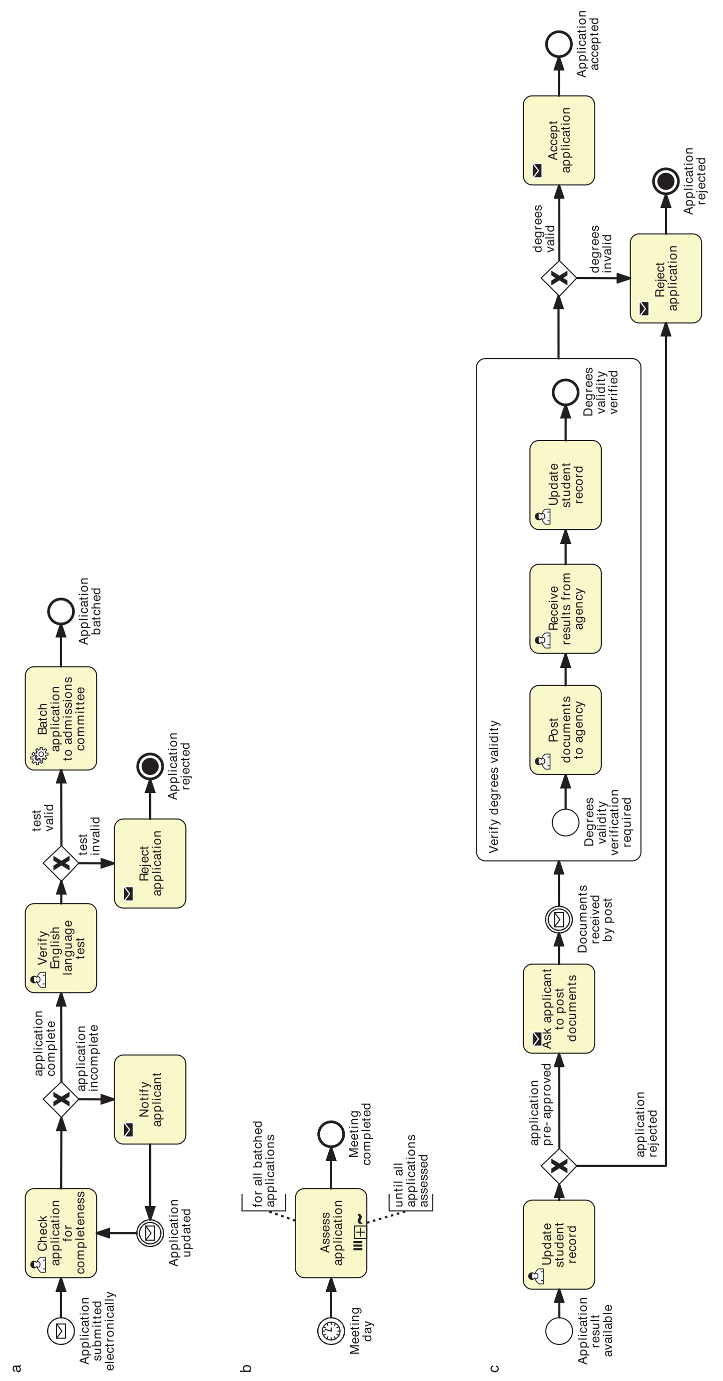


Fig. 9.7 Admission process: the initial (**a**) and final (**c**) assessments can be automated in a BPMS; the assessment by the committee (**b**) is a manual process outside the scope of the BPMS

BPMN, a process that starts with an untyped event indicates that instances of this process are explicitly started by a BPMS user, in our case a clerk at the admissions office. This process initiation is called *explicit instantiation* as opposed to *implicit instantiation* where process instances are triggered automatically by the event type indicated in the start event, e.g. an incoming message or a timer.

Finally, when the process is mostly or entirely made up of unordered manual tasks, automation does not provide any benefit, and is often even unfeasible. Take for example the post-production process in the screen industry. Activities like "Hold spotting session", where director and composer watch the picture to decide the music cues, "Conduct negative cutting", where the camera negative is manually cut and spliced to match the final edit specified by the film editor, and "Compose soundtrack", can hardly be coordinated by a BPMS, since there is no strict order for their execution and each of them may be repeated multiple times. As a rule of thumb, a process whose tasks are performed in an ad-hoc manner, without any predictable order, is not suitable for automation via a production BPMS. In this case, a case handling or ad-hoc workflow system would be more appropriate.

Exercise 9.8 Consider the final part of the prescription fulfillment process described in Exercise 1.6:

> Once the prescription passes the insurance check, it is assigned to a technician who collects the drugs from the shelves and puts them in a bag with the prescription stapled to it. After the technician has filled a given prescription, the bag is passed to the pharmacist who double-checks that the prescription has been filled correctly. After this quality check, the pharmacist seals the bag and puts it in the pick-up area. When a customer arrives to pick up their prescription, a technician retrieves the prescription and asks the customer for their co-payment or for the full payment in case the drugs in the prescription are not covered by the customer's insurance policy.

One way of modeling this fragment is by defining the following tasks: "Check insurance", "Collect drugs from shelves", "Check quality", "Collect payment" (triggered by the arrival of the customer), and finally "Retrieve prescription bag". Assume the pharmacy system automates the prescription fulfillment process. Identify the type of each task and if there are any manual tasks, specify how these can be linked to the pharmacy system.

There are other modeling elements, besides manual tasks, that are relevant at a conceptual level but cannot be interpreted by a BPMS. These are physical data objects and data stores, messages bearing physical objects and text annotations. Pools and lanes are also used at a conceptual level only. In fact, as we have seen, pools and lanes are often used to capture coarse-grained resource assignments, e.g. activity "Confirm order" is done within the sales department. When it comes to execution, we need to define resource assignments for each task and capturing this information via dedicated lanes (potentially one for each task) will just make the diagram too cluttered. Electronic data stores are also not directly interpreted by a BPMS, as the BPMS assumes the existence of dedicated services that can access these data stores, e.g. an inventory information service that can access the warehouse DB. So the BPMS will interface with these services rather than directly with the data stores. Also, the state of a data object indicated in the object's label, e.g. "Purchase order [confirmed]", cannot be interpreted as such by a BPMS. Later we will show how to explicitly represent object states so that they can be interpreted by a BPMS.

Some BPMSs tolerate the presence of these non-executable elements too. If this is the case, we suggest to leave these elements in. Especially pools, lanes, message flows bearing electronic objects, electronic data stores and annotations will guide us in the specification of some execution properties. For example, the Sales lane in the order fulfillment model tells us that the participant to be assigned task "Confirm order" has to be from the sales department. Other BPMS modeling tools do not support these elements, so it is not even possible to represent them in the diagram.

Exercise 9.9 Consider the loan assessment model that you obtained in Exercise 9.7. Identify the modeling elements that cannot be interpreted by a BPMS.

9.4.3 Complete the Process Model

Once we have established the automation boundaries of the process and reviewed manual tasks, we need to check that our process model is *complete*. Often business-oriented process models neglect certain information because modelers deem it is not relevant for the specific modeling purpose, they assume it is common knowledge, or simply, they are not aware of it. It may be fine to neglect this information in a business-oriented model, depending on the application scenario. However, information that is not relevant in a business-oriented model may be highly relevant for a process model to be executed.

A typical example is when the process model focuses on the "sunny-day" scenario and neglects all negative situations that may arise during the execution of the process, working under the assumption that everything will work well. As we saw in Chap. 4, the order fulfillment process is indeed subjected to a number of exceptions. For example, it may be aborted if the materials required to manufacture the product are not available at the suppliers or if the customer sends an order cancellation. So we need to make sure that all exceptions are handled using appropriate exception handlers. For example, if the order cancellation is received after the product has been shipped or after the payment has been received, we also have to compensate for these activities by returning the product and reimbursing the customer. Another exception that is commonly neglected is that representing an activity that cannot complete. What happens if the customer's address is never received? Or if the ERP module for checking the stock availability does not respond? We cannot assume that the other party will always respond or that a system will always be functional. Similarly, we cannot assume that activities always lead to a positive outcome. For example, an order may not always be confirmed.

You may be surprised how rarely exceptions are modeled in a business-oriented process in practice. Thus, in the majority of cases, such a model will require to be completed with these aspects before being executed.

In this step, we also need to specify all *electronic data objects* that are required as input and output by the tasks of our process. For instance, in Fig. 9.6 there is no input data object to task "Request raw materials from Supplier 1(2)", though this task does need the list of raw materials to be ordered. Another example is task "Check stock availability". This task uses the Purchase order as input (to obtain the code of the product to be looked up in the Warehouse DB) but does not produce any output data to store the results of the search. However, without this information, the subsequent XOR-split cannot determine which branch to take (we can now see why this is called *data-based XOR-split*). If you have not noticed the absence of these data objects so far, it is probably because you assumed their existence. This is fine in a business-oriented model where only aspects relevant to the specific modeling purpose are documented, but not in an executable model, where an engine has to run the model. So, make sure each activity has the required input and output electronic data objects. The principle is that every data object needed by the BPMS engine to pass control between activities and to take decisions must be modeled.

The completed order fulfillment example, including exception handlers and data objects that are relevant for execution, is shown in Fig. 9.8.[1]

Exercise 9.10 Take the loan assessment model that you obtained in Exercise 9.6 after incorporating the revisions from Exercise 9.7. Complete this model with control-flow and data-flow aspects relevant for automation. For simplicity, you may disregard the modeling elements that are not interpretable by a BPMS.

9.4.4 Bring the Process Model to an Adequate Granularity Level

There is not necessarily a one-to-one mapping between the tasks in a business-oriented model and those in the corresponding executable model. Indeed, we should keep in mind that a BPMS is intended to coordinate and manage handovers of work between multiple resources (human or non-human). Accordingly, two or more consecutive tasks assigned to the same resource are candidates for *aggregation*. If this was the case, the BPMS would not add value between these two tasks because it would not manage any handover. All it would do is to interfere in the work of a given resource. For example, a sequence of user tasks "Enter customer name", "Enter customer policy number" and "Enter damage details", such that all three tasks will be performed by the same claims handler, should be aggregated into a single user task "Enter claim".

[1]The content of the sub-processes and some of the elements that cannot be interpreted by a BPMS have been omitted for simplicity.

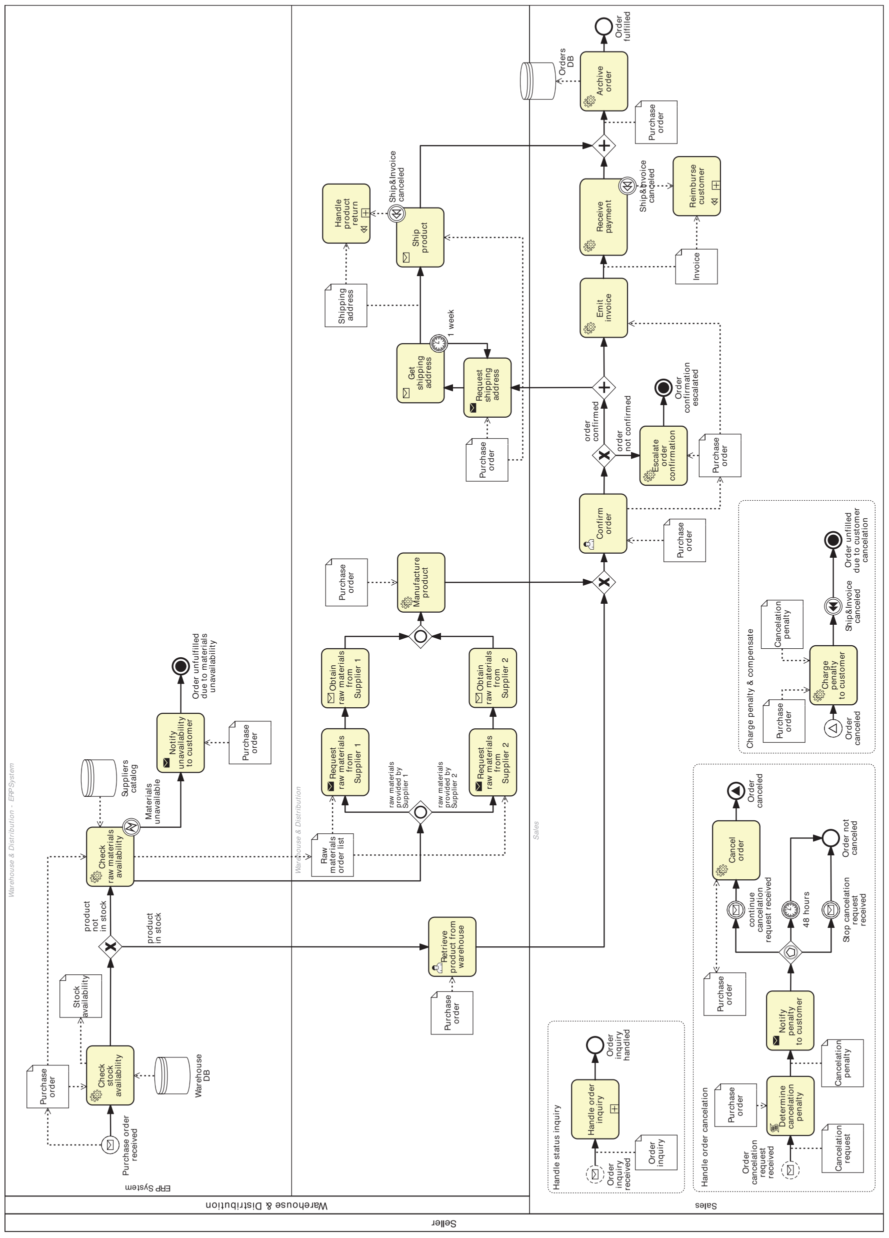


Fig. 9.8 The order fulfillment model of Fig. 9.6, completed with control-flow and data-flow aspects relevant for automation

There are some cases, though, where we may actually need to keep consecutive tasks separate, despite they are performed by the same resource. For example, in

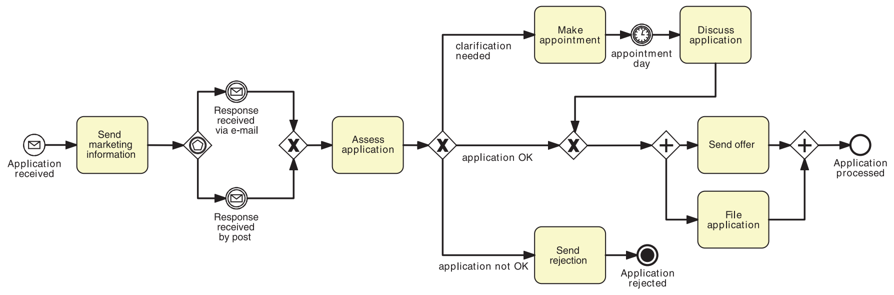


Fig. 9.9 The sales process of a B2B service provider

Fig. 9.7c we have three user tasks within sub-process "Verify degrees validity": "Post documents to agency", "Receive results from agency" and "Update student record". While these may be performed by the same admin clerk, we do want to keep track of when each task has been completed, for the sake of monitoring the progress of the application and manage potential exceptions. For example, if the results are not received within a given timeframe, we can handle this delay by adding an exception handler to task "Receive results from agency".

Exercise 9.11 Are there activities that can be aggregated in the model obtained in Exercise 9.10? Hint: candidate tasks for aggregation may not necessarily be consecutive due to a sub-optimal order of tasks in the business-oriented model. In this case, you need to resequence the tasks first (see Sect. 8.2.3).

In a similar vein, if an activity requires more than one resource to be performed, it is too *coarse-grained*, so we should disaggregate it into more fine-grained tasks such that these can be assigned to different resources. For example, an activity "Enter and approve money transfer" is likely to be performed by two different participants even if they have the same role, in order to enforce separation of duties: first a financial officer enters the order, then a different financial officer approves it.

Exercise 9.12 Figure 9.9 shows the model for the sales process of a business-to-business (B2B) service provider. The process starts when an application is received from a potential client. The client is then sent information about the available services and a response is awaited either via e-mail or postal mail. When the response is received, the next action is decided upon. Either an appointment can be made with the client to discuss the service options in person, or the application is accepted or rejected right away. If the application is accepted, an offer is sent to the client and at the same time the application is filed. If it is rejected, the client is sent a thank-you note and let go. If an appointment has to be made, this is done and at the time of the appointment, the application is discussed with the client. Then the process continues as if the application had been accepted right away.

1. Identify the type of each task and find ways of linking the manual tasks to a BPMS.

2. Remove elements that cannot be interpreted by a BPMS.
3. Complete the model with control-flow and data aspects required for execution.
4. Bring the resulting model to a granularity level that is adequate for execution.

Acknowledgement This exercise is adapted from a similar exercise developed by Remco Dijkman, Eindhoven University of Technology.

9.4.5 Specify Execution Properties

In the last step, we need to specify *how* each model element is effectively implemented by the BPMS. For example, take the first service task of our revised order fulfillment example: "Check stock availability". Saying this task requires the purchase order as input to contact the warehouse ERP system is not enough. We need to specify the service provided by the ERP system to check stock levels and its location in the network, the product information in the purchase order that is required by this service (i.e. the format of the input object) and the information produced by the service (i.e. the format of the output object). These implementation details are called *execution properties*. More, specifically, these are:

- Process variables, messages, signals and errors
- Task and event variables and their mappings to process variables
- Service details for service, send and receive tasks, and for message and signal events
- Code snippets for script tasks
- Participant assignment rules and user interface structure for user tasks
- Task, event and sequence flow expressions
- BPMS-specific properties

These properties do not have a graphical representation in the BPMN diagram, but are stored in the BPMN 2.0 interchange format. The BPMN interchange format is a textual representation of a BPMN model in XML format. It is intended to support the interchange of BPMN models between tools and also to serve as input to a BPMN execution engine. BPMN modeling tools provide a visual interface to edit most of these non-graphical properties, so most of the times you will not need to write the XML directly. Still, you will need to understand standard Web technology, especially XML, XML Schema (XSD), and be familiar with the notion of (Web) service, to be able to implement a process model. This section assumes that you have basic knowledge of these technologies. We provide pointers to further readings on these technologies in Sect. 9.8.

Figure 9.10 shows the structure of the BPMN format. It consists of a list of elements, where some are optional (those with a dashed border) and others are mandatory (those with solid borders). The process element is mandatory and stores information about the process model. This consists of electronic data objects, events, tasks and flows. The elements outside the process are reusable components needed

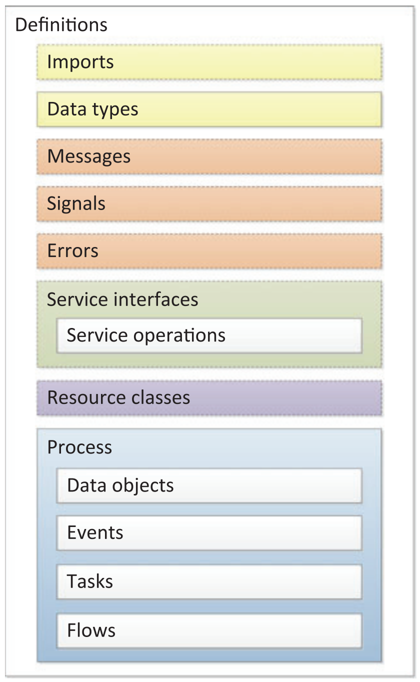


Fig. 9.10 Structure of the BPMN format

by the various process elements, like message definitions and service interfaces which are used by service, send and receive tasks, and by message and signal events. With reference to this structure, let us now go through each of the execution properties above.

Process Variables, Messages, Signals and Errors Process variables are managed by the BPMS engine to allow data exchange between process elements. Each electronic data object, e.g. the purchase order in the order fulfillment process, represents a process variable. The lifetime of a process variable is confined to the life of the process instance in which the variable is created, and is only visible to the process level in which it is defined and to all its sub-processes. This means that a variable defined in a sub-process is not visible in the parent process.

We need to assign a *data type* to each process variable in order for a BPMS to be able to interpret and manipulate these variables. In BPMN, the type of each process variable is specified as an XSD type. The type of a variable can be *simple* or *complex*. Simple types are strings, integers, doubles (numbers containing decimals), booleans, dates, times, etc., and are already defined in the XSD specification. For example, the object Stock availability can be represented as a process variable of type integer (representing the number of available units of a product). Complex types are hierarchical compositions of other types. A complex type can be used for example to represent a business document, such as a purchase order or an invoice. Figure 9.11a shows the complex type of the purchase order, called purchaseOrderType, while a)

```
<complexType name="purchaseOrderType">
    <sequence>
        <element name="order">
            <complexType>
                <sequence>
                    <element name="orderNumber" type="integer"/>
                    <element name="orderDate" type="date"/>
                    <element name="status" type="string"/>
                    <element name="currency" type="string"/>
                    <element name="productCode" type="string"/>
                    <element name="quantity" type="integer"/>
                </sequence>
            </complexType>
        </element>
        <element name="customer">
            <complexType>
                <element name="name" type="string"/>
                <element name="surname" type="string"/>
                <element name="address">
                    <complexType>
                        <sequence>
                            <element name="street" type="string"/>
                            <element name="city" type="string"/>
                            <element name="state" type="string"/>
                            <element name="postCode" type="string"/>
                            <element name="country" type="string"/>
                        </sequence>
                    </complexType>
                </element>
                <element name="phone" type="string"/>
                <element name="fax" type="string"/>
            </complexType>
        </element>
    </sequence>
</complexType>
```

b)

```
<purchaseOrder>
    <order>
        <orderNumber>15664</orderNumber>
        <orderDate>2012-10-23</orderDate>
        <status>confirmed</status>
        <currency>EUR</currency>
        <productCode>345-EAR</productCode>
        <quantity>10</quantity>
    </order>
    <customer>
        <name>John</name>
        <surname>Brown</surname>
        <address>
            <street>8 George St</street>
            <city>Brisbane</city>
            <state>Queensland</state>
            <postCode>4000</postCode>
            <country>Australia</country>
        </address>
        <phone>+61 7 3240 0010</phone>
        <fax>+61 7 3221 0412</fax>
    </customer>
</purchaseOrder>
```

Fig. 9.11 The XSD describing the purchase order (**a**) and one of its instances (**b**)

Fig. 9.11b is the XML representation of a particular purchase order instance at run-time. From the type definition we can see that a purchase order contains a sequence of two elements:

- Order, to store the order information (order number, order date, status, currency, product code and quantity), and
- Customer, to store the customer information (name, surname, address, phone and fax)

The data fields order, customer and address are complex types so that they can contain sub-elements. Also, observe the field status within order: this is used to capture the state of the purchase order, e.g. “confirmed”.

Similar to process variables, we also need to assign data types to each message, signal and error used in the process model. For the messages, we can look at the existing message flows in the diagram and define one data type for each uniquely labeled message flow. So for example if we have two message flows labeled purchase order, they will obviously take the same type purchaseOrderType. If message flows are not modeled, we can look at the send, receive and service tasks, and at the message events present in the diagram in order to understand what messages to define. For signals and errors we have to look at the signal and error events that we have defined in the diagram. While for a signal, the data type describes the content of the signal being broadcasted or listened to, for an error the data type defines what information is carried with the error. For example, if it is a system error, we can use this to specify the error message returned by the system. In addition, we need to assign an *error code* to each error. This code uniquely identifies an error within the process model, so that a catching error event can be related to a throwing error event.

Task and Event Variables Besides the above data elements, we need to define the internal variables of each task, called *data inputs* and *data outputs* in BPMN. Data inputs and outputs act as interfaces between a task and its input and output data objects. They also need to refer to an XSD type defining their structure, but different from process variables, they are only visible within the task (or sub-process) in which they are defined. Data inputs capture data that is required by the task to be executed; data outputs capture data that is produced by the task upon completion. Thus, data inputs are populated with the content of input data objects while data outputs are used to populate the content of output data objects. For example, we need a data input for task "Check stock availability" in order to store the content of the purchase order. Thus, the type of this data input must match that of the input object, i.e. purchaseOrderType. Similarly, the data output must be of type integer to store the number of items in stock, so that this information can be copied into stock availability upon task completion.

The mapping between data objects and task data inputs/outputs is defined via the task *data associations*. Data associations can also be used to define complex data assignments beyond one-to-one mappings. For example, consider task "Manufacture product". The service invoked by this task only requires the order product code and quantity in order to start the manufacturing of the product. Thus, we can use a data association to extract the product code and quantity from the input purchase order, and populate a data input containing two sub-elements of types string, respectively, integer. In most cases, the BPMS will automatically create all the tedious data mappings between data objects and tasks. For example, for the case above all we need to do is to select the sub-elements of the purchase order we want to use as input to "Manufacture product", and the BPMS will create the required data inputs and their mappings for this task. BPMN relies on XPATH 1.0 as the default language for expressing data assignments like the one above. However, other languages can be used like Java Universal Expression Language (UEL) or Groovy. The choice depends on the BPMS adopted. For example, Activiti supports UEL, Bonita Open Solution and Camunda Fox support Groovy while BizAgi's BPM Suite supports its own expression language.

Similar to tasks, events that transmit or receive data, i.e. message, signal and error events, also have internal variables. Specifically, the catching version of these events has one data output only, to store the content of the event being caught (e.g. an incoming message), whereas the throwing version has one data input only, to store the content of the event being thrown (e.g. an error). Thus, we also need to assign these data inputs and outputs a type that has to match that of the message, signal or error associated with the event. For example, the start catching message event “Purchase order received” in the order fulfillment example uses a data output to store the purchase order message once this has been received. Thus, this data output must match the type of the incoming message, which is precisely purchaseOrderType. In turn, the output object must have the same type as the output data, to contain the purchase order.

The complex types for all data elements of the process can be defined directly in the BPMN model or imported from an external document (see Fig. 9.10).

Service Tasks Once we have defined the types of all data elements, and mapped task and event data inputs and outputs to these types, we have to specify how tasks and events have to be implemented. For service tasks we need to specify how to communicate with the external application that will execute the task. Be it a complex system or a simple application, from the perspective of the BPMS all that is required is that the external application provides a *service interface* that the service task can use. A service interface contains one or more *service operations*, each describing a particular way of interacting with a given service. For example, a service for retrieving inventory information provides two operations: one to check the current stock levels and one to check the stock forecast for a given product (based on product code or name). An operation can either be *in-out* or *in-only*. In an in-out operation (also called *synchronous* operation), the service expects a request message and replies with a response message once the operation has been completed, or optionally with an error message if something goes wrong. For example, the service invoked by task “Check raw materials availability” receives stock availability information as input message and replies with a list of raw materials to be ordered as output message. Alternatively, if the service experiences an exception (e.g. the suppliers catalog is unreachable), it replies with an error message which triggers the boundary error event of this task so that the relative exception handler can be performed.[2] Conversely, in an in-only operation (also called *asynchronous* operation), the service expects a request message but will not reply with a response message. For example, task “Archive order” notifies an archival service of the purchase order to be archived, however, the process does not wait for an archival confirmation.

Each message of a service operation needs to reference a message in the BPMN model, so that it can be assigned a data type. For instance, the request and the response messages to interact with the inventory service have data type purchaseOrderType, respectively, XSD integer. For each interface, we also need to specify how this is concretely implemented, i.e. what communication protocols are used by

[2]Note that there is no throwing end error event inside “Check raw materials availability” since the catching error event is triggered by the receipt of an error message by the service task. The ability to link error messages with error events is a common feature of BPMSs.

the service and where the service is located in the network. By default, BPMN uses Web service technology to implement service interfaces, and relies on WSDL 2.0 to specify this information. In practice, this corresponds to defining one or more external WSDL documents and importing them into our BPMN model. Once again, other implementations are possible, e.g. one could implement a service interface via Java remote procedure call or plain XML over HTTP.

After defining the service interfaces for our process, we need to associate each service task with a service operation defined in a service interface. Based on the type of the operation (in-out or in-only), we then need to define a single data input that must match the type of the request message in the referenced service operation, and optionally a single data output that must match the type of the response message in the operation. The BPMS engine will copy the task data input to the request message and send it out to the service, and once the response message has been received, will copy the content of this message to the task data output.

Send and Receive Tasks, Message and Signal Events Send and receive tasks work similarly. A send task is a special case of the service task: it sends a message to an external service using its data input, but there is no response. An example is task “Notify unavailability to customer”. A receive task waits for an incoming message and uses its data output to store the message content. Task “Get shipping address” is an example of this. Both task types need to reference an in-only service operation where the message is defined. However, for the receive, the message being received is seen as a request coming from an external service requester. Thus, in this case the process itself acts as the service provider.

A receive task can also be used to receive the response of an asynchronous service which has previously been invoked with a send task. This is the case of tasks “Request shipping address” and “Get shipping address”. The asynchronous service is provided by the customer. Accordingly, in the send task the seller’s process acts as the service requester sending a request message to the customer. In the receive task the roles get swapped: the seller acts as the service provider to receive the response message from the customer. This pattern is used for long-running interactions, where the response may arrive after a while. The drawback of using a synchronous service task in place of a send-receive is that this task would block the process to wait for the response message. This is not the case in Fig. 9.8, where the send and receive tasks are in parallel to “Emit invoice” which may thus be performed in-between.

Message and signal events work exactly like send and receive tasks. For signal events, it is assumed that the service being contacted has publish-subscribe capabilities, e.g. a Web service for subscribing to RSS feeds.

Script Tasks For script tasks, we need to provide the snippet of code that will be executed by the BPMS. This code can be written in a programming language such as JavaScript or Groovy. BPMN does not prescribe the use of a specific programming language so the choice depends on the BPMS used. The task data inputs store the parameters for invoking the script while the data outputs store the results of executing the script. For example, for task “Determine cancellation penalty” we can define a script that extracts the order date and the cancellation request date from two data inputs mapped to the input objects purchase order and cancellation request, uses this information to compute a penalty of €15 for each day past the order date, and copies this value to the data output.

User Tasks For each user task we need to specify the rules for assigning work items of this task to process participants at runtime, the technology to communicate with participants and the details of the user interface to use. Moreover, like for any other task, we need to define data inputs to pass information to the participant, and data outputs to receive the results.

Process participants that can be assigned user tasks are called *potential owners* in BPMN. A potential owner is a member of a resource class. In the context of user tasks, a resource class identifies a static list of *participants* sharing certain characteristics, e.g. holding the same role or belonging to the same department or unit. An example of resource class for the order fulfillment process is order clerk, which groups all participants holding this role within the sales department of the seller organization. Note that these resource classes are unrelated to pools and lanes, which are only notational elements in a business-oriented process model. A resource class can be further characterized by one or more *resource parameters*, where a parameter has a name and a data type. For example, we can define two parameters product and region of type string to indicate the particular products an order clerk works with, and the region they work in.

Once we have defined all required resource classes and optionally their parameters, we can assign each user task to one or more resource classes based on an expression. For example, we can express that work items of task “Confirm order” have to be assigned to all participants of type Order clerk who deal with the particular product being ordered and work in the same region as the customer. For this, we can define an XPATH expression that selects all members of Order clerk whose properties product and region are equal to the product code, respectively, country contained in the purchase order.

We also need to specify the implementation technology used to offer the work item to the selected participant(s). This entails aspects such as how to reach the participant (e.g. via email or worklist notification), how to render the content of the task data inputs on screen (e.g. via one or more web forms organized through a particular screenflow), and the strategy to assign the work item to a single participant out of those satisfying the assignment expression (e.g. assign it to the order clerk with the shortest queue or randomly). The configuration of these aspects, as well as the association of participants to resource classes is dependent on the specific BPMS being used.

Task, Event and Sequence Flow Expressions Finally, we need to write expressions for the various attributes of tasks and events, and for the sequence flows bearing conditions. For instance, in a loop task we need to write a boolean expression that implements the textual annotation indicating the loop condition (e.g. “until response approved”). This boolean expression will determine when will the loop task be repeated. This expression can be defined over data elements, e.g. it can be an XPATH expression that extracts the value of the boolean element "approved" from a Response object. We can also use *instance attributes* inside these expressions. These are variables that vary by instance at execution. An example is *loop count*, which counts the number of iterations for a loop task. For the timer event we need to specify an expression to capture the temporal event informally expressed by its label (e.g. "Friday afternoon"). Here we have three options: we can either provide a temporal expression in the form of a precise date or time, a relative duration, or a repeating interval. Once again, these expressions can be linked to data elements and instance properties so as to be resolved dynamically at execution. For example, we can set an order confirmation timeout based on the number of line items in an order. Finally, we need to write a boolean expression to capture the condition attached to each sequence flow following an (X)OR-split. For example, condition "product in stock" after the first XOR-split in the order fulfillment example can be implemented as an XPATH expression that checks whether the value of variable stock availability is at least equal to the product quantity contained in the purchase order. There is no need to assign an expression to a default sequence flow, since this arc will be taken by the BPMS engine if the expressions assigned to all other arcs emanating from the same (X)OR-split evaluate to false.

BPMS-Specific Properties Strictly speaking, the only BPMS-specific properties that we have to configure in order to make a process model executable are those of user tasks. In practice, however, we will likely need to link our executable process with the enterprise system of our organization. This is called *system binding*. Luckily, BPMSs offer a range of predefined service task extensions, called *service adapters*service adapter (or *service connectors*), to implement common system binding functions in a convenient way. Examples of such binding functions include: performing a database lookup, sending an email notification, posting a message to Twitter or setting an event in Google Calendar, reading or writing a file and adding a customer in a CRM system. Each adapter comes with a list of parameters that we need to configure. However, BPMSs provide wizards with capabilities to auto-discover some of the parameter values. For instance, to use a database lookup we need to provide the type of the database server (e.g. MySQL, Oracle DB) and the URL where the server can be reached, the schema to be accessed, the SQL query to run and the credentials of the user authorized to run the query.

Coming back to our example, instead of implementing task "Check stock availability" as a service task, which assumes the existence of an inventory information service at the seller, we could implement this task with a generic database lookup adapter, provided we know what to search for and where. Similarly, we could implement the tasks for communicating with the customer like "Notify unavailability to customer" and "Request shipping address" as email adapters, so that we do not need to implement a dedicated email service in our organization. The number and variety of adapters that a BPMS provides largely contribute to increasing the value of the product over competing solutions.

Exercise 9.13 Consider the loan assessment process model that you obtained in Exercise 9.11. The loan application contains these data fields:

- Applicant information:
  - Identity information (name, surname, ...)
  - Contact information (home phone, cell phone, ...)
  - Current address (street name and number, city, ...)
  - Previous address (as above plus duration of stay)
  - Financial information (job details, bank details)
- Reference information (identity, contact, address, relation to applicant)
- Property information (property type, address, purchasing price)
- Loan information (amount, number of years, start date, interest type: variable/fixed)
- Application identifier
- Submission date and time
- Revision date and time
- Administration information (a section to be compiled by the loan provider):
  - Status (a string to keep track of the state of the application, with predefined values: "incomplete", "complete", "assessed", "rejected", "canceled", "approved")
  - Comments on status (optional, e.g. used to explain the reasons for rejection)
  - Eligibility (a boolean to store whether or not the applicant is eligible for a loan)
  - Loan officer identifier
  - Insurance quote required (a boolean to store whether or not a home insurance quote is sought)

The credit history report contains these data fields:

- Report identifier
- Financial officer identifier
- Reference to a loan application
- Applicant's credit information:
  - Loan applications made in the last five years (loan type: household/personal/domestic, amount, duration, interest rate)
  - Overdue credit accounts (credit type, default amount, duration, interest rate)
  - Current credit card information (provider: Visa, Mastercard, ..., start date, end date, interest rate)
  - Public record information (optional, if any):
    - Court judgments information
    - Bankruptcy information
- Credit assessment (a string with predefined values: AAA, AA, A, BBB, BB, B unrated).

The risk assessment contains the following data fields:

- Assessment identifier
- Reference to a loan application
- Reference to a credit history report

- Risk weight (an integer from 0 to 100)

The property appraisal contains the following data fields:

- Appraisal identifier
- Reference to a loan application
- Property appraiser identifier
- Property information (property type, address)
- Value of three surrounding properties with similar characteristics
- Estimated property market value
- Comments on property (optional, to note serious flaws the property may have)

The agreement summary contains the following data fields:

- Reference to a loan application
- Conditions agreed (a boolean indicating if the applicant agreed with the loan conditions)
- Repayment agreed (a boolean indicating if the applicant agreed with the repayment schedule)
- Link to digitized copy of the repayment agreement

The loan provider offers a website where applicants can submit and revise loan applications online, track the progress of their applications and if required, cancel applications in progress. This website implements an underlying Web service with which the loan assessment process interacts. In practice, this service acts as the applicant from the perspective of the loan assessment process. For example, if the applicant submits a new loan application through the website, this service wraps this application into a message and sends it to the BPMS engine of the loan provider, which in turn starts a new instance of the loan assessment process. If the loan assessment process sends an application for review to this service, the service presents this information to the applicant via the loan provider's website.

Further, the loan assessment process interacts with an internal service for assessing loan risks. This service determines a risk weight which is proportional to the credit assessment contained in the credit history report, on the basis of the applicable risk rules read from a database (the interaction between the service and the database is transparent to the BPMS). The risk assessment service returns a risk assessment containing an identifier (freshly generated), a reference to the loan application and one to the credit history report (both extracted from the credit history report), and the risk weight.

Based on the above information, specify the execution properties for the elements of this process model. You are not required to define the actual XSD type of each data element, nor to specify the actual Groovy scripts or XPATH expressions. All you need to do is to identify what properties have to be specified, i.e. what data inputs and outputs, service interfaces, operations, messages and errors are required, and determine their data type in relation to that of process variables. For example, a data input may map to a process variable or to a data field within this. For scripts, you need to define via task data inputs and outputs what data is required by the script, what data is produced and how the input data is transformed into the output one. For example, based on the value of a data field in a process variable, a script may write a particular value in the data field of another process variable. Similarly, for each user task, you need to identify what information is presented to the task performer, and how the data output is obtained. Finally, you need to explain how each expression can be evaluated on the basis of data fields within process variables (e.g. to implement the condition of a sequence flow), or constant values like a date (e.g. to implement a timer event).

9.4.6 The Last Mile

Now that you have become familiar with what is required to turn a process model executable, the last step for you is to take a process model and implement it using a BPMS of your choice (e.g. Activiti, Bonita Open Solution, Bizagi's BPM Suite, YAWL). The landscape of BPMSs and their specificities evolves continuously. We can identify three categories of BPMSs with respect to their support for BPMN:

1. **Pure BPMN** These tools have been designed from the ground up to support BPMN natively. They follow the specification "to the letter" though they might not fully support it. Examples are Activiti and Camunda Fox.
2. **Adapted BPMN** These tools use a BPMN skin but rely on an internal representation to execute the process model. They can import and sometimes also export in the BPMN 2.0 format. They typically predate BPMN and evolved from previous versions to support the specification. Examples are Bizagi's BPM Suite and Bonita Open Solution.
3. **Non BPMN** There is finally a general category of BPMSs which use their own proprietary language and semantics. These tools do not support BPMN. Examples are BPMOne from Perceptive Software and YAWL.

At the time of writing, most of the BPMSs supporting BPMN in one way or another, still do not cover all aspects of the specification that are relevant for execution. For example, elements like message boundary events, compensation events and non-interrupting events are hardly supported. So concretely we have to give up on one or more of these elements depending of the BPMS that we adopt. One expects though that support for executable BPMN will increase over time.

This section illustrated how to design executable BPMN models in a vendor-independent manner. The book's website (http://fundamentals-of-bpm.org) provides tutorial notes showing how to configure an executable process model for concrete BPMSs.

Exercise 9.14 Based on the execution properties that you specified in Exercise 9.13, implement the loan assessment process using a BPMS of your choice.

9.5 Recap

In this chapter we focused on a specific type of process-aware information system, namely Business Process Management Systems (BPMSs). We discussed the architecture of a BPMS and its main components: the execution engine, the process modeling tool and the process model repository, the administration and monitoring tools and the execution logs, as well as the external services that can be invoked.

There are many reasons for considering process automation. First, it provides workload reduction in terms of coordination: work is assigned to process participants or software services as soon as it is available. Second, it offers integration flexibility. Processes can be changed with significantly less effort as compared to legacy systems, provided they are explicitly represented via process models. Third, the execution in a BPMS generates valuable data on how processes are executed, including performance-relevant data. Finally, BPMSs improve the quality of process execution as they directly enforce rules such as separation of duties.

Introducing BPMSs poses various challenges. Technical challenges arise from the fact that many applications that have to be integrated are typically not designed as open systems with transparent interfaces. Beyond that, organizational challenges are rooted in the fact that BPMSs directly interfere with how people do their job. This fact calls for sensitive change management.

Finally, we presented a method for transforming business-oriented process models into executable specifications, so that they can be interpreted by a BPMS. First, we need to identify the type of each process task (automated, manual or user) and review manual tasks to find, whenever it is possible, a way to link these to the BPMS. Next, we need to complete the process model by specifying all control-flow and data aspects that are relevant for execution and bridge the diverging level of granularity between a business-oriented process model and its executable counterpart. Finally, we need to specify a number of execution properties for each model element. Some of these properties, like for user tasks, are vendor-specific and so will vary depending on the specific BPMS that we decide to adopt.

9.6 Solutions to Exercises

Solution 9.1 There are three current work items:

1. Case #1,220: Determine Proper Piece of Equipment
2. Case #1,230: Determine Proper Piece of Equipment
3. Case #1,240: Complete Equipment Rental Request

Solution 9.2

- The execution engine would be unable to determine to allocate work items to resources on the basis of a process model alone, when it would only cover control-flow information.