2 Modeling Workflows

2.1 Workflow Concepts

The success of a workflow system stands or falls on the quality of the workflows put into it. This book therefore devotes considerable attention to the modeling and analysis of workflows. In this chapter, we shall limit ourselves initially to the process itself. As a tool, we use Petri nets. With their help, we can represent a process in a straightforward way. We can also use them to analyze these processes. We shall go into this aspect more extensively in chapter 4. Before any of this, we should first examine some of the concepts introduced in chapter 1 in more detail.

2.1.1 The case
The primary objective of a workflow system is to deal with cases. Examples of cases include an insurance claim, a mortgage application, a tax return, an order, or a patient in a hospital. Similar cases belong to the same case type. In principle, such cases are dealt with in the same way.

Each case has a unique identity. This makes it possible to refer to the case in question. A case has a limited lifetime. Consider, for example, an insurance claim. This case begins at the moment when the claim is submitted and disappears from the workflow system at the point when the processing of the claim has been completed. Between the appearance and disappearance of a case, it always has a particular state. This state consists of three elements: (1) the values of the relevant case attributes', (2) the conditions that have been fulfilled; and (3) the content of the case.

A range of variables can be associated with each case. These case attributes are used to manage it. Thanks to them it is, for example, possible to indicate that a task may—under certain conditions—be omitted.
When handling an insurance claim, we may use the case attribute "estimated claim value." Based upon the value of this variable, the workflow system can decide whether or not to activate the "send assessor" task. Note that the value of a case attribute may change as the case progresses.

We cannot use a case attribute to see how far a case has progressed. To do this, we use conditions. These are used to determine which tasks have been carried out, and which still remain to be performed. Examples of conditions include "order accepted," "application refused," and "under consideration." We can also regard a condition as a requirement that must be met before a particular task may be carried out. Only once all the conditions for a task within a particular case have been met can that task be performed. For any given case, it is at all times clear which conditions have been met and which not. We can also use the word phase instead of condition. This, however, is confusing when several conditions have been met: the case would be in more than one phase simultaneously.

In general, the workflow system does not contain details about the content of the case, only those of its attributes and conditions. The content is contained in documents, files, archives, and/or databases that are not managed by the workflow management system.

2.1.2 The task
The term task already has been mentioned extensively. It refers to one of the most important concepts in this book. By identifying tasks, it is possible to structure workflows. A task is a logical unit of work. It is indivisible and thus is always carried out in full. If anything goes wrong during the performance of a task, then we must return to the beginning of the entire task. In this respect, we refer to a rollback. However, the indivisible nature of a task depends upon the context within which it is defined. A task which is contracted out by a client to a supplier is regarded as "atomic" (indivisible) by the former. This does not have to be the case for the supplier, though: he may well split the task set into smaller ones.

Typing a letter, assessing a valuation report, filing a complaint, stamping a document, and checking personal data are all examples of tasks. We can differentiate between manual, automatic and semi-automatic tasks. A manual task is entirely performed by one or more people, with-
A task refers to a generic piece of work, and not to the performance of an activity for one specific case. In order to avoid confusion between the task itself and the performance of that task as part of a particular case, we use the terms work item and activity. A work item is the combination of a case and a task which is just about to be carried out. A work item is created as soon as the state of a case allows it. We thus can regard a work item as an actual piece of work which may be carried out. The term activity refers to the actual performance of a work item. As soon as work begins upon the work item, it becomes an activity. Note that, unlike a task, both a work item and an activity are linked with a specific case. Figure 2.1 shows this diagrammatically.

2.1.3 The process
The way in which a particular category of cases should be carried out is described by the relevant process. This indicates which tasks need to be carried out. It also shows the order in which this should be done. We can also regard a process as a procedure for a particular case type. In general, many different cases are handled using a single process. It therefore is possible to enable a specific treatment based upon the attributes of a certain case. For example, it may be that one task in the process is only
performed on some of the cases. The order in which the tasks are performed may also vary depending upon the properties of the case. Conditions are used to decide which order is followed. In essence, a process is therefore constructed from tasks and conditions.

It is possible to make use of previously defined processes as part of another process. So, in addition to tasks and conditions, a process may also consist of (zero or more) subprocesses. Each of the subprocesses again consists of tasks, conditions, and possibly even further subprocesses. By explicitly identifying and separately describing subprocesses, frequently occurring ones can be used repeatedly. In this way, complex processes can also be structured hierarchically. At the highest level of process description, we see a limited number of subprocesses. By examining one or more of these we can, as it were, "zoom in" on particular sections of the process.

The lifecycle of a case is defined by a process. Because each case has a finite lifetime, with a clear beginning and end, it is important that the process also conforms with this. So each process also has a beginning and an end, which respectively mark the appearance and completion of a case.

2.1.4 Routing

The lifecycle of a case is laid down in the process. In this respect, we refer to the\textit{ routing} of the case. Routing along particular branches determines which tasks need to be performed (and in which order). In routing cases, we make use of four basic constructions:

• The simplest form of routing is the \textit{sequential} execution of tasks. In other words, they are carried out one after the other. There is usually also a clear dependency between them. For example, the result of one task is input to the next.

• If two tasks can be performed simultaneously, or in any order, then we refer to \textit{parallel} routing. In this case, there are two tasks which both need to be performed without the result of one affecting the other. The two tasks are initiated using an \textit{AND-split} and later resynchronized using an \textit{AND-join}.

• We refer to \textit{selective routing} when there is a choice between two or more tasks. This choice may depend upon the specific properties of the case, as recorded in the relevant case attributes. Such a choice between alternatives is also known as an \textit{OR-split}. The alternative paths are
reunited using an OR-join. As well as selective routing, we also use the terms alternative or conditional routing.

- In the ideal situation, a task is carried out no more than once per case. Sometimes, however, it is necessary to perform a particular task several times. Consider, for example, a task which needs to be repeated until the result of the subsequent "check" task is satisfactory. We call this form of routing iteration.

We shall return to these four forms of routing in more detail later.

2.1.5 Enactment
A work item assignment can only be carried out once the state of the case in question allows it. But actual performance of such an assignment often requires more than this alone. If it has to be carried out by a person, he must first take the assignment from his "in tray" before an activity can begin. In other words, the work item is worked on only once the employee has taken the initiative. In such a case we refer to triggering: the work item is triggered by a resource (in the example, an employee). However, other forms of triggering are possible: an external event (for example, the arrival of an EDI message) or reaching a particular time (for example, the generation of a list of orders at six o'clock). We thus distinguish between three types of triggers: (1) a resource initiative, (2) an external event, and (3) a time signal. Work items which must always be carried out immediately—without the intervention of external stimuli—do not require a trigger.

The concepts summarized above are the central themes of this chapter. We thus shall focus mainly upon the modeling of the processes which underlie the workflows. In the next chapter, we shall turn our attention to the allocation of work items, the arrangement of the organizational structure, and specific staff skills. In chapter 4, we shall see how we can analyze the workflows modeled.

2.2 Petri Nets

Unlike many other publications on workflow management, this book takes a formal approach based upon an established formalism for the modeling and analysis of processes—Petri nets. The use of such a formal concept has a number of major advantages. In the first place, it forces
precise definition. Ambiguities, uncertainties, and contradictions are thus prevented, in contrast to many informal diagramming techniques. Secondly, the formalism can be used to argue about processes. It thus becomes possible, for example, to establish certain patterns. This is closely linked with the fact that a formalism often enables the use of a number of analytical techniques (those for analyzing performance, for instance, as well as those for verifying logical properties). As we shall see later, it becomes possible to check whether a case is successfully completed after a period of time. There thus are various good reasons to opt for a formal method. Before we depict the concepts listed earlier in this chapter within Petri nets, it is important to know something about this technique. For the sake of completeness, we shall go deeper into them than is strictly necessary for the purposes of workflow management.

Petri nets were devised in 1962 by Carl Adam Petri as a tool for modeling and analyzing processes. One of the strengths of this tool is the fact that it enables processes to be described graphically. Later, we shall see that we can use it to present workflow processes in an accessible way. Despite the fact that Petri nets are graphical, they have a strong mathematical basis. Unlike many other schematic techniques, they are entirely formalized. Thanks to this formal basis, it is often possible to make strong statements about the properties of the process being modeled. There are also several analysis techniques and tools available which can be applied to analyze a given Petri net.

Over the years, the model proposed by Carl Adam Petri has been extended in many different ways. Thanks to these, it is possible to model complex processes in an accessible way. Initially, however, we shall confine ourselves to the classic Petri net as devised by Petri himself.

2.2.1 Classical Petri nets

A Petri net consists of places and transitions. We indicate a place using a circle. A transition is shown as a rectangle. Figure 2.2 shows a simple Petri net, consisting of three places (claim, under Consideration, and ready) and three transitions (record, pay, and send_letter). This network models the process for dealing with an insurance claim. Arriving at the place claim, it is first recorded, after which either a payment is made or a letter sent explaining the reasons for rejection.
Places and transitions in a Petri net can be linked by means of a directed arc. In figure 2.2, for example, the place *claim* and the transition *record* are linked by an arrow pointing from the former to the latter. There are two types of arcs: those that run from a place to a transition and those that run from a transition to a place. Arcs from a place to a place or a transition to a transition are not possible.

Based upon the arcs, we can determine the input places of a transition. A place $p$ is an input place for a transition $t$ if—and only if—there is a directed arc running from $p$ to $t$. Similarly, we can determine the output places of a transition. A place $p$ is an output place for a transition $t$ if—and only if—there is a directed arc running from $t$ to $p$. As it happens, in figure 2.2 each transition precisely has one input and one output place.

Places may contain tokens. These are indicated using black dots. In figure 2.2 the place *claim* contains three tokens. The structure of a Petri net is fixed; however, the distribution of its tokens among the places can change. The transition *record* can thus take tokens from the *claim* input place and put them in *under Consideration*. We call this the firing of the transition *record*. Because the firing of transitions is subject to strict rules, we shall first introduce a number of terms.

The state of a Petri net is indicated by the distribution of tokens amongst its places. We can describe the state illustrated in figure 2.2 using the vector $(3,0,0)$. In other words, there are three tokens in *claim*, none in *under Consideration*, and none in *ready*.

A transition may only fire if it is enabled. This occurs when there is at least one token at each of its input places. The transitions are then, as it were, "loaded": ready to fire. In figure 2.2, the transition *record* is enabled. The other two are not.
A transition may fire from the moment it is enabled. As it fires, one token is removed from each input place and one token added to each output place. In other words, the moment it fires, a transition consumes tokens from the input place and produces tokens for the output place. Figure 2.3 shows the effect of firing the transition record. Its result is that one token is transferred from the place claim to the place under Consideration. We can also describe the new situation using the vector (2,1,0).

Once record has fired, a situation arises in which three transitions are enabled. The transition record can fire again because there is at least one token in claim, and the transitions pay and send letter can fire because there is a token in under Consideration. In this situation, it is not possible to tell which transition will fire first. If we assume—for the sake of convenience—that it is the transition pay which fires, then the state illustrated in figure 2.4 will be reached.

Note that the transition send_letter, which was enabled before firing, is no longer enabled. The transition record is still enabled and will therefore fire. Eventually, after a total of six firings, the Petri net will reach the state (0,0,3). That is, a state with three tokens in the place ready. In this state, no further firing is possible.
Transitions are the *active* components in a Petri net. By firing a transition, the process being modeled shifts from one state to another. A transition therefore often represents an event, an operation, a transformation, or a transportation. The places in a Petri net are *passive*, in the sense that they cannot change the network’s state. A place usually represents a medium, buffer, geographical location, (sub)state, phase, or condition. Tokens often indicate objects. These can be physical ones, but also objects representing information. In the network considered above, each token represents an insurance claim.

In the Petri net shown in figure 2.2, it is possible for several cases to be in progress simultaneously. If the transition *record* fires twice in succession, then there will be at least two tokens in the place *under consideration*. If, for some reason, we wish to limit the number of cases which can be under consideration at the same time to a maximum of one, then we can modify the Petri net as shown in figure 2.5. The additional place *free* ensures that the transition *record* is blocked as soon as a claim goes under consideration.
In the initial state depicted, \textit{record} is enabled because there is at least one token at each of the input places. Once transition \textit{record} has fired, the state is such that \textit{record} is no longer enabled, but the other two transitions are. Once one of these has fired, there is again a token in the place \textit{free}. Only at this point is \textit{record} again enabled. By adding the place \textit{free}, the maximum number of cases that can be under consideration at any time has indeed been limited to one. If we wish to limit the number of cases in progress at any time to a maximum of \textit{n}, then we can model this simply by placing \textit{n} tokens in the place \textit{free} at the start.

Using Petri nets, we can also describe processes that are repetitive in nature. Figure 2.6 shows how we can model the cyclical activity of a set of traffic lights.

The traffic lights' three possible settings are illustrated by three places: \textit{red}, \textit{yellow}, and \textit{green}. The three possible light changes are shown by the transitions \textit{rg}, \textit{gy}, and \textit{jr}. Imagine now that we want to model the traffic lights at the crossing of two one-way streets. In this case, we require two sets of traffic lights that interact in such a way that one of the two is always red. Obviously, the Petri net shown in figure 2.6 needs to be duplicated. Each set of lights is modeled using three places and three transitions. This, however, is not sufficient, because it does not exclude unsafe situations. We therefore add an extra place \textit{x}, which ensures that one of the two sets of lights is always at red (see figure 2.7).
Figure 2.7  
Two sets of traffic lights

When both traffic lights are red, there is a token in the place $x$. As one set of lights changes to green, the token is removed from $x$ and so the other set is blocked. Only when both sets of lights are again red is the other able to change to green once. In chapter 4, we use an analytical technique to show that the traffic lights do indeed operate safely.

2.2.2 High-level Petri nets

Because Petri nets are graphical, they are easily accessible and easy to use. They also have a strong mathematical basis and there are many analytical techniques available for them. In chapter 4, we shall see that we can use these techniques to analyze workflows. Despite this strength, the classic Petri net has shortcomings in many practical situations. It becomes too large and inaccessible, or it is not possible to model a particular activity. This is why the classic Petri net has been extended in many ways. Thanks to these extensions, it is possible to model complex situations in a structured and accessible way. In this section we shall focus upon the three most important extensions: (a) color extension, (b) time extension, and (c) hierarchical extension. We call Petri nets extended with color, time, and hierarchy high-level Petri nets. Because a complete description of high-level Petri nets would go too far, we shall confine ourselves to those aspects that are important in the context of workflow management.

(a) The color extension Tokens are used to model a whole range of things. In one model they can represent insurance claims, in another the state of traffic lights. However, in the classic Petri net it is impossible to
distinguish between two tokens: two in the same place are by definition indistinguishable. In general, this is an undesirable situation. In the case of two insurance claims, for example, we want to incorporate the separate characteristics of the two claims in the model. We want to include such things as the nature of the claim, the policy number, the name of the policyholder, and the assessed value of the claim. In order to enable the coupling of an object's characteristics with the corresponding token, the classic Petri net is extended using "color." This extension ensures that each token is provided with a value or color. A token representing a particular car will, for instance, have a value which makes it possible to identify its make, registration number, year of manufacture, color, and owner. We can notate a possible value for such a token as follows: \[\text{[brand: 'BMW'; registration: 'fj 144 NFX'; year: '1995'; color: 'red'; owner: 'Johnson']}\].

Because each token has a value, we can distinguish different tokens from one another. By "valuing" tokens, they are—as it were—given different colors.

A firing transition produces tokens that are based upon the values of those consumed during firing. The value of a produced token therefore may depend upon those of consumed ones. Unlike in the classic Petri net, the number of tokens produced is also variable: the number of tokens produced is determined by the values of those consumed.

To illustrate this, we shall use a process for dealing with technical faults in a product department. Every time a fault occurs—for example, a jammed machine—it is categorized by the department's mechanic. The fault can often be put right as it is being categorized. If this is not the case, then a repair takes place. After this has been done, a test is carried out, with three possible results: (1) the fault has been solved; (2) a further repair is required; or (3) the faulty component must be replaced. This process is modeled in figure 2.8 using a Petri net.

A token in the place fault means that a fault has occurred which needs to be dealt with. For each token in fault, the transition categorize will fire precisely once. During each firing precisely one token will be produced, in either the place solved or the place nr (needs repair). In contrast with the classic Petri net, it is now possible for an output place not to receive a token. During the execution of transition categorize, a choice is now made based upon the information available. As a result of this choice, the
fault is either regarded as solved or a repair is carried out. The token in the place *fault* has a value in which the relevant properties of the fault are recorded (for example: the nature of the fault, the identity of the non-functioning component, its location code, and fault history). If a repair is required, then the transition *repair* will fire, bringing the token to place *nt*, followed by the firing of transition *test*. The transition *test* produces precisely one token, which appears in one of the three output places. The relevant information about the fault is always retained in the value of the token in question.

In a color-extended Petri net, we can set conditions for the values of the tokens to be consumed. If this is the case, then a transition is only enabled once there is a token at each of the input places and the *preconditions* have been met. A transition's precondition is a logical requirement connected with the values of the tokens to be consumed. In the Petri net illustrated in figure 2.8, we could for example add the following precondition to the transition *categorize*: "The value of the token to be consumed from the place *fault* must contain a valid location code." The consequence of this precondition is that faults without a valid location code are not categorized; they remain in the place *fault* and are never consumed by the transition *categorize*.

We can also use a precondition to "synchronize" tokens. By this we mean that a transition only fires if a particular combination of tokens can be consumed. We use the transition *assemble*, illustrated in figure 2.9, to illustrate this.

Based upon a production order, the transition *assemble* takes a chassis, an engine, and four wheels and produces a car. (This is the first example
we have seen in which more than one arrow leads from an input point to a transition. In this case, there must be at least four tokens in wheel before assemble can be enabled. The number of incoming arrows thus shows how many tokens there must be at the input point from which they come. When a transition fires, the number of tokens consumed is equal to the number of incoming arrows.) When the transition assemble fires, tokens are not taken at random from the input places. For example, the four wheels must be of the same type, the engine must fit the chassis, the wheel diameter must suit the chassis and the engine power, and so on. Tokens thus are only taken from the input places in certain combinations. This is determined by means of a precondition.

The result of the color extension is that, in contrast to the classic Petri net, the graphic representation no longer contains all the information. For each transition, the following factors must be specified:

• Whether there is a precondition. If there is a precondition, then this must be defined precisely.
• The number of tokens produced per output place during each firing. This number may depend upon the values of the tokens consumed.
• The values of the tokens produced. This, too, may depend upon the values of the tokens consumed.

Depending upon the objective for which the Petri net has been produced, the transitions are specified by a piece of text, a few lines of pseudo-code, a formal specification, or a subroutine in a programming language.
(b) The time extension

Given a process modeled as a Petri net, we often want to be able to make statements about its expected performance. If we produce a model of the traffic lights at a road junction, then we are probably also interested in the number of vehicles that this junction can handle per hour. If we model the production process in a car factory, then we also want to know the expected completion time and the capacity required. To be able to answer these questions, it is necessary to include pertinent information about the timing of a process in the model. However, the classic Petri net does not allow the modeling of "time." Even with the color extension, it is still difficult to model the timing of a process. Therefore, this classic Petri net is also extended with time.

Using this time extension, tokens receive a *timestamp* as well as a value. This indicates the time from which the token is available. A token with timestamp 14 thus is available for consumption by a transition only from moment 14. A transition is enabled only at the moment when each of the tokens to be consumed has a timestamp equal or prior to the current time. In other words, the enabling time of a transition is the earliest moment at which its input places contain sufficient available tokens. Tokens are consumed on a FIFO (first-in, first-out) basis. The token with the earliest timestamp thus is the first to be consumed. Furthermore, it is the transition with the earliest enabling time that fires first. If there is more than one transition with the same enabling time, a non-deterministic choice is made. Moreover, the firing of one transition may affect the enabling time of another.

If a transition fires and tokens are produced, then each of these is given a timestamp equal to or later than the time of firing. The tokens produced thus are given a *delay* that is determined by the firing transition. The timestamp of a produced token is equal to the time of firing plus this delay. The length of the delay may depend upon the value of the tokens consumed. However, it is also possible that the delay has a fixed value (for example, 0) or that the delay is decided at random. Firing itself is instantaneous and takes no time.

To illustrate the time extension, we can use the example of the two sets of traffic lights, which must not simultaneously be at green or yellow. At moment 0 both sets are at red. As we can see in figure 2.10, the time-stamps of the tokens in the places red1, x, and red2 are 0.
Chapter 2

Figure 2.10
The two sets of traffic lights with time

The enabling time of the transition $rg1$ is also 0, the maximum of the
timestamps of the tokens in $red1$ and $x$. The enabling time of $rg1$ is
also 0. There hence exists a nondeterministic choice between $rg1$ and
$rg1$. Let us assume that $rg1$ fires. The transition $rg1$ consumes the two
tokens from the input places and produces one token for the place $green1$
with a delay of 25 time units. In figure 2.10, each delay is shown as a
label linked to an arrow emerging from a transition. (If the delays were
dependent upon the values of the tokens consumed, this would no longer
be possible.) After the firing of $rg1$, there is a token in $green1$ with a
time stamp of 25, and $gy1$ is the only enabled transition. The transition
$gy1$ thus will fire at moment 25 and produce a token at $yellow1$ with a
timestamp equal to $25 + 5 = 30$. At moment 30, the transition $yr1$ will
fire. During this firing, $yr1$ produces a token for $red1$ with a delay of 30
and a token for $x$ without delay. As a result of the firing, $rg1$ has an
enabling time of 60 and $rg2$ an enabling time of 30. Therefore transition
$rg2$ now fires. By adding time to the model, we thus have not only
specified the timing of the various phases, but also forced the traffic lights
to change to green alternately.

(c) The hierarchical extension Although we can already describe very
complex processes using the color and time extensions, usually the re-
resulting Petri net still will not provide a proper reflection of the process
being modeled. Because the modeling of such a process results in a single,
extensive network, any structure is lost. We do not observe the hierar-
The process "solve fault" contains one subprocess: "repair.

*chical structure* in the process being modeled by the Petri net. The hierarchical extension therefore ensures that it becomes possible to add structure to the Petri net model.

In order to structure a Petri net hierarchically, we introduce a new "building block": a double-bordered square. We call this element a *process*. It represents a subnetwork comprising places, transitions, arcs, and subprocesses. Because a process can be constructed from subprocesses that in turn also can be constructed from (further) subprocesses, it is possible to structure a complex process hierarchically. In order to illustrate this, we shall refine the process modeled in figure 2.8. This refinement concerns the activity *repair*. We no longer wish to regard repair as a single, indivisible action, but as a subprocess consisting of the following steps: (1) start, (2) trace, (3) change, and (4) end. Moreover, there is never more than one fault under repair at a given point in time. To model this refinement, we replace the transition *repair* with a subprocess consisting of four transitions and four places—see figure 2.11.

In figure 2.11, we can see clearly that a process can take two forms: (1) as a subprocess within a hierarchically superior process (the double-bordered square), and (2) as the definition of the process (a summary of
the elements from which the process is constructed). We find the meaning of a process constructed from subprocesses by replacing each of those subprocesses with the appropriate definition. The process solve fault illustrated in figure 2.11 is thus in fact a Petri net consisting of six transitions and nine places.

By using (sub)processes, we can structure a Petri net hierarchically, using either a top-down or a bottom-up approach. The top-down approach begins at the highest level, with processes increasingly being broken down into subprocesses until, at the lowest level, these consist only of transitions and places. Repeated decomposition results in a hierarchical description. The bottom-up approach works in the opposite direction. It begins at the lowest level. First, the most elementary components are described in detail. These elements (subprocesses) are then combined into larger processes. Repeated composition eventually results in a description of the entire process.

When modeling complex processes, a hierarchical method of description is often an absolute necessity. Only by dividing the main process into ever-smaller subprocesses can we overcome its complexity. In this respect, we refer to the divide-and-conquer strategy. However, the identification of subprocesses has yet another important advantage. It enables us to reuse previously defined processes. If a particular subprocess recurs several times, one definition used repeatedly will suffice. The reuse of (sub)processes often makes it possible to model a complex process more quickly.

In this section, we have studied the three most important types of extensions: (a) the color extension, (b) the time extension, and (c) the hierarchical extension. We call Petri nets which incorporate these extensions high-level Petri nets. In the remainder of this book, we shall use the high-level net to model and analyze processes in the context of workflow management.

2.3 Mapping Workflow Concepts onto Petri Nets

The time has now come to illustrate the concepts described earlier—the case, task, condition, process, trigger, and so on—using the Petri net technique.
2.3.1  The process

Using a process in a workflow management system, we can indicate in which way a particular category of cases should be handled. The process defines which tasks need to be carried out. As well as information about the tasks to be performed, a process also contains information about conditions. In this way, it defines the order in which the tasks need to be carried out. It is also possible to use previously defined processes within a larger process. Thus process may also consist of more than one sub-process, as well as tasks and conditions. It therefore is obvious to specify a process using a Petri net. This network should have one "entrance" (a place without incoming arcs) and one "exit" (a place without outgoing arcs). We show conditions as places and tasks as transitions. This also is obvious, because transitions are the active components in a Petri net, and places its passive components.

In order to specify a process using a Petri net, we shall examine a process for handling complaints. An incoming complaint first is recorded. Then the client who has complained and the department affected by the complaint are contacted. The client is approached for more information. The department is informed of the complaint and may be asked for its initial reaction. These two tasks may be performed in parallel—that is, simultaneously or in any order. After this, the data are gathered and a decision is taken. Depending upon the decision, either a compensation payment is made or a letter is sent. Finally, the complaint is filed. Figure 2.12 shows how we can illustrate the process just described using a Petri net.

Each of the tasks record, contact_client, contact_department, pay, and file is modeled using a transition. The assessment of a complaint is modeled using two transitions: positive and negative. The transition positive corresponds with a positive decision; the transition negative corresponds with a negative decision. (Later we shall see how this task can also be modeled using just one transition.) The places start and end correspond with the beginning and end of the process being modeled. The other places correspond with conditions that are or are not met by every case in progress. The conditions play two important roles: on the one hand they ensure that the tasks proceed in the correct order, and on the other hand that the state of the case can be established. The place c8, for example, ensures that a complaint is filed only once it has been fully
Figure 2.12
The process "handle complaint" modeled as a Petri net

dealt with. It also corresponds with the state that exists between a complaint being fully handled and its filing.

From the above, it should be more or less clear that a case is represented by one or more tokens. Cases thus are illustrated using tokens. In figure 2.12, the token in the place _start_ shows the presence of a case. Once _record_ has fired, there are two tokens—one at _c1_, one at _c2_—that represent the same case. As a case is being handled, the number of its tokens thus may fluctuate. The number of tokens that represent a particular case is equal to the number of its conditions that have been met. Once there is a token in _end_, the case has been completed. In principle, each process should fulfil two requirements: (1) it should at any time be possible to reach—by performing a number of tasks—a state in which there is a token in _end_; and (2) when there is a token in _end_, all the others should have disappeared. These two requirements ensure that every case that begins at the place _start_ will eventually be completed properly. Note that it is not possible to have a token in _end_ while there remain tasks still to be performed. The minimum requirements just mentioned, which every process must meet, can be checked effectively using standard Petri net tools.

The state of a case is not determined solely by the conditions that have been met; to steer it, the case may have one or more attributes. For these, it seems obvious to use the color extension. The value of a token contains information about the attributes of the case in question. We shall go into this in more detail later.
Tokens that correspond with particular cases are kept strictly separate (by the workflow management system). We can translate this into Petri net modeling in two ways. Because tokens belonging to different cases cannot influence one another, we can produce a separate copy of the Petri net for each case. Each thus has its own process, as illustrated in figure 2.12. However we can also use just one Petri net by making use of the color extension. Thanks to this, we can provide each token with a value from which it is possible to identify the case to which the token refers. This is shown diagrammatically in figure 2.13.

The state of the Petri net illustrated here indicates that there are currently five cases in progress. Case 1 has almost been completed, whereas case 5 is still at the start state. In order to ensure that the token belonging to different cases do not get "mixed up," each transition is provided with a precondition that states that only tokens from the same case may be consumed at any one firing. If the transition $collect$ in the situation shown in figure 2.13 now fires, this precondition will ensure that the two tokens for case 3 are consumed.

Figure 2.12 shows a nonhierarchical process. However it goes without saying that a process may be constructed from subprocesses. To illustrate this, we can for example combine the first four tasks ($record$, $contact\_client$, $contact\_department$, and $collect$) into a single subprocess called $phasisel$. Figure 2.14 shows how the corresponding Petri net would look, with two levels.
2.3.2 Routing

Tasks may be optional. That is, there may be tasks that only need to be carried out for a number of cases. The order in which tasks are performed may also vary from case to case. By routing a case along a number of tasks, we can determine which tasks need to be carried out (and in what order). As indicated earlier, four basic constructions for routing are recognized. For each of these, we shall show the corresponding Petri net modeling.

(a) **Sequential routing** We refer to the sequential performance of tasks when these have to be carried out one after another. If two tasks need to be carried out sequentially, there usually is a clear interdependence between them. For example, the result of the first is required in order to perform the second. In a Petri net, this form of routing is modeled by linking the two tasks using a place. Figure 2.15 shows an example of sequential routing.
The task that corresponds with the transition \textit{task2} is only performed once the task corresponding with transition \textit{task1} has been completed. This is enforced by place \textit{c2}, which corresponds with the condition that must apply before \textit{task2} can be carried out.

(b) Parallel routing If more than one task can be carried out at the same time or in any order, then we refer to parallel routing. If we confine ourselves to the situation with two tasks, \textit{task1} and \textit{task2}, then there are three possibilities: both tasks can be performed simultaneously; \textit{task1} can be carried out first, then \textit{task2}; or \textit{task2} can be first, followed by \textit{task1}. Figure 2.16 illustrates how we can model this situation using a Petri net. In order to enable the parallel execution of \textit{task1} and \textit{task2} in the case corresponding with the token in \textit{c1}, we begin with a so-called \textit{AND-split}. This is a task added so as to allow more than one task to be managed at the same time. In figure 2.16, the transition \textit{t1} is the equivalent of an AND-split. It fires when there is a token in \textit{c1}, and produces one token in each \textit{c2} and \textit{c3}. Once condition \textit{c2} has been met for a particular case, \textit{task1} can be carried out. Once condition \textit{c3} has been met, \textit{task2} can be carried out. Firing \textit{t1} thus enables the performance of two tasks. We also say that \textit{task1} and \textit{task2} can be carried out in parallel. Only when both
have been performed can transition \( t_2 \) fire. It is the equivalent of an \textit{AND-join}: a task added to synchronize two or more parallel flows. Only when a particular case has fulfilled both condition \( c_4 \) and condition \( c_5 \) this task can be performed.

In figure 2.16, we have had to insert two tasks, \( t_1 \) and \( t_2 \), to model the \textit{AND-split} and the \textit{AND-join}. We call such "artificial" additions management tasks, because they do not correspond with a recognizable piece of work. Thanks to them, we can carry out task1 and task2 in parallel. However, it is also possible for tasks such as \( t_1 \) and \( t_2 \) to correspond with an actual piece of work. In figure 2.12, for example, the task \textit{record} corresponds with an \textit{AND-split}. The task \textit{collect} corresponds with an \textit{AND-join}.

In a business process in which cases are carried out entirely manually (without the aid of a workflow system), sequential routing is often the rule due to, for example, physical limitations. For instance, the tasks in a particular case must be carried out one after the other because the accompanying document can only be in one place at a time. By introducing a workflow system, such limitations are largely eliminated. Tasks that previously had to be carried out sequentially can now be done in parallel. This can often achieve enormous time savings. Allowing parallel routing thus is clearly of major significance in the success of a workflow system.

(c) Selective routing A process lays down the routing for a specific type of case. But there may be differences in routing between individual cases. Consider, for example, a process for dealing with insurance claims. Depending upon the specific circumstances of a claim, a particular route will be selected. The task \textit{send_assessor}, for example, is not carried out for small claims. We refer to such cases as selective routing. This involves a choice between two or more tasks. Figure 2.17 shows an example modeled in terms of a Petri net.

Once a case fulfils condition \( c_1 \), either \( t_{11} \) or \( t_{12} \) fires. If it is the former, then task1 is enabled. If it is the latter, then it is task2 that is enabled. Thus there is a choice between the two tasks. We call the network consisting of transitions \( t_{11} \) and \( t_{12} \) and places \( c_2 \) and \( c_3 \) an \textit{OR-split}. Once one of the tasks has been performed, the \textit{OR-join} ensures that a token appears in \( c_6 \). In this case, the OR-join is modeled using a network
consisting of two places ($c_4$ and $c_5$) and two transitions ($t_{21}$ and $t_{22}$). So the OR-split selects one of the two alternative streams and the OR-join brings them back together. In figure 2.17, we have explicitly modeled the OR-split and the OR-join by adding two small networks. This is necessary when we want to show the OR-split and OR-join as explicit management tasks. However, it is also possible to model them implicitly, as shown in figure 2.18.

When a case fulfills condition $c_1$, either task1 or task2 will be carried out. So this is another example of selective routing. If we look at the way in which the OR-join is modeled in the two previous figures, we notice little difference. Obviously, therefore, an OR-join can be modeled using several arrows leading into the same place. In the case of the OR-split, though, there is a difference. In figure 2.17, a choice is made at the moment when there is a token in $c_1$ (that is, when a case fulfills condition $c_1$ in figure 2.18, the choice comes later. Which of the two branches is
actually selected is decided only at the moment when either \textit{task1} or \textit{task2} has to be carried out. This may appear to be only a subtle difference, but in fact the distinction between the OR-splits in figures 2.17 and 2.18 can be of crucial importance.

Let us assume, for example, that \textit{task1} corresponds with the processing of a valuation report, and that \textit{task2} has to be carried out if that report is not delivered within a given time. In this context, the model provided using the construction in figure 2.18 is excellent. When the token is in \textit{c1}, two subsequent events are possible: either the report arrives and \textit{task1} is carried out, or it is late and \textit{task2} is carried out. The decision about which task to perform is delayed until either the report arrives or a fixed period of time has elapsed. In figure 2.17, however, the decision must be taken immediately. \textit{If t11}, for example, fires, then it is no longer possible to carry out \textit{task2}. Later on, we shall show some larger examples in which the \textit{moment} the choice is made is of great significance.

Thus far, we have (automatically) assumed that the choice between two alternatives is nondeterministic. In other words, we have not explained how the choice between \textit{task1} and \textit{task2} is made, because—as far the process is concerned—it does not matter which task is performed: the selection is left to the environment of the workflow system. In most cases, however, the decision is made best according to the specific properties of the case. Depending upon the values of the case attributes (that is, the case's management parameters), we want to be able to choose between the alternatives. Figure 2.19 shows how we can model this situation.

Based upon the case attributes, transition \textit{t1} in figure 2.19 produces a token for either \textit{c2} or \textit{c3} (but not for both). In this case, therefore, we make use of color extension to enable a choice to be made in transition \textit{t1}. Using the attributes of the case in question, the decision rule in \textit{t1} determines which task should be performed. In doing so, we assume that all the relevant attributes of this case are contained in the value of the token in \textit{c1}. In the case of parallel routing, however, there may be more than one token assigned to the same case. Because the attributes concern the entire case, these tokens must have identical values. In other words, there must never be two tokens assigned to the same case but with different values. In order to enforce this, we must ensure that a change to a case attribute caused by the performance of a task updates the value of \textit{every} token pertaining to that case.
We thus can regard a case attribute as information that can be inspected and revised by every task relevant to that case. In theory, the broad nature of a case attribute can be modeled explicitly by linking each transition with a common place. This place always contains one token whose value corresponds with those of the case attributes. Because illustrating this common place makes the process diagrams confusing, for the sake of convenience we shall omit it.

In figure 2.19, the number of tokens produced in each of the output places of $t_l$ is variable (0 or 1). A choice is made based upon the value (case attributes) of the token in $c_l$ and the decision rule in $t_l$. However, we can also produce this choice by using two transitions containing the appropriate preconditions. Recall that a precondition is based on the colors of the tokens to be consumed and acts like a transition guard. Figure 2.20 shows how this is possible.

The precondition in transition $t_{11}$ corresponds with the requirements that need to be met to justify the choice for task1. The precondition in $t_{12}$ determines when task2 should be selected. If the precondition in $t_{11}$ is the negation of the precondition in $t_{12}$, then each token in $c_l$ will result in a deterministic choice for either task1 and task2. In this case, therefore, the OR-splits in figures 2.19 and 2.20 are equivalent.

Because constructions such as the AND-split, AND-join, OR-split and OR-join occur frequently, we use a special notation to illustrate them. This is shown in figure 2.21.

We represent an AND-split by using the symbol $\Box$ on the output side. This indicates that a token must be produced for each of the output places under all circumstances.
We represent an AND-join by using the symbol ♦ on the input side. This indicates that the task being modeled can only take place once there is a token at each of the input places. From figure 2.21, we can see that both the AND-split and the AND-join correspond with a "normal transition" like those encountered in the classic Petri net.

We represent an OR-split by using the symbol ♣ on the output side. This indicates that a token must be produced for precisely one of the output places. As we saw earlier, we can model this in two ways. In the rest of this chapter, we shall use only the first of these.

We represent an OR-join by using the symbol ♣ on the input side.

We can use the following technique to remember the difference between the AND and OR symbol. When, in principle, the arrows enter or leave the same large triangle, it is an AND. Otherwise, it is an OR.

The symbol ♣ on the output side indicates a mixture of an AND-split and an OR-split. In this case one or more tokens will be produced, depending upon the value of the case attributes. Figure 2.21 shows two ways of using this mixed form in a Petri net.

(d) Iterative routing The last form of routing is the repeated execution of a particular task. Ideally, a task will be performed only once per case. In certain situations, however, it is necessary to apply iterative routing. For example, when a certain task needs to be repeated until the results of a subsequent test prove positive. Figure 2.22 shows how we can model iterative routing.
Figure 2.21
Notation method for common constructions

**Notation**

- **AND-split**
  
- **AND-join**
  
- **OR-split**
  
- **OR-join**
  
- **AND/OR-split**

**Meaning**

- **(possible) preconditions**

- **decision rule**

- **(possible) preconditions**

- **decision rule**

*Modeling Workflows*
Taking the case corresponding with the token in c1, we see that task1 and task2 are performed successively. Once task2 has been completed, OR-split t determines whether or not it needs to be performed once again. Once task2 has been carried out one or more times, the case moves on to task3. Task2 must be carried out at least once between task1 and task3.

Figure 2.22 assumes that task2 must be performed at least once ("repeat ... until ..."). If this is not the case, the construction illustrated in figure 2.23 applies ("while ... do ...").

Immediately upon completion of task1, OR-split t determines whether or not task2 needs to be carried out. It now becomes possible for task1 to be followed directly by task3.

In both examples, there exists an OR-split that makes its decision based upon the current values of the case attributes. Note that the two constructions illustrated correspond with the familiar "repeat ... until ..." and "while ... do ..." constructions that appear in many programming languages.

Example Using the example described in the previous chapter, we can now illustrate the concepts defined thus far. The example concerns an
Modeling Workflows

insurance company's process for dealing with claims. Chapter 1 identifies sixteen tasks in this process. In chapter 1 we did not yet introduce the Petri net tool to model workflow processes. Therefore, we used an "ad hoc" notation technique to illustrate the routing. Now, however, we can show the process "properly," as shown in figure 2.24. But before looking at that diagram, try to model the process yourself.

For the sake of convenience, the conditions which are used to route the cases correctly are given "symbolic" names. In practice, however, symbolic names are of no use. For example, we could more appropriately call condition \( c_7 \) accepted. Conditions \( c_1 \) and \( c_{20} \) have a special role: \( c_1 \) represents the start of the process and \( c_{20} \) its end. Note that the "informal" diagram in chapter 1 and figure 2.24 do closely resemble one another. The major difference between the two is that the conditions are explicitly named in figure 2.24. As a result, we can describe the state of a case.

2.3.3 Enactment

A process is a collection of tasks, conditions, subprocesses, and their relationships with one another. As we have seen, we can describe a process using a Petri net. Conditions are depicted using places and tasks using transitions. To simplify the representation of a process in terms of a Petri net, we have defined a method of notating a number of typical constructions. (See figure 2.21.)

A process is designed to deal with a particular category of cases, and so may handle many individual cases. A task is not specific to a particular case. However, when a case is being carried out by a process, tasks are performed for that specific case. In order to avoid confusion between a task as such and its performance on a specific case, we have introduced the terms work item and activity. A work item is the combination of a case and a task which is ready to be carried out. The term activity refers to the actual performance of a work item. At the point when a work item is actually being worked on, it is transformed into an activity. Note that, unlike a task, both a work item and an activity are linked to a specific case. The distinction between (1) a task, (2) a work item, and (3) an activity becomes clear as soon as we translate them into Petri net terms. A task corresponds with one or more transitions, a work item with a transition being enabled, and an activity with the firing of a transition.
Figure 2.24
The process for dealing with insurance claims
Transitions in a Petri net are "eager." In other words, they fire as soon as they are enabled. As we have just established, the enabling of a transition corresponds with a work item. For an assignment to be carried out, however, more is often required than simply the relevant case having the right state. If it is to be carried out by a person, she must first take it from her "in tray" before an activity begins. In other words, the work item is only carried out once the employee has taken the initiative. This is why we recognized the existence of triggering. Certain work items can only be transformed into an activity once they have been triggered.

We differentiate between three types of triggers: (1) a resource initiative (such as an employee taking a work item from her in tray); (2) an external event (such as the arrival of an EDI message); and (3) a time signal (such as the generation of a list of orders at six o'clock). Work items that must always be carried out immediately, without the intervention of a resource, do not need a trigger. We can illustrate in a Petri net which form of triggering applies. Tasks triggered by a resource are shown using a wide, downward-facing arrow. Those triggered by an external event have an envelope symbol. And those that are time dependent have a clock symbol. Figure 2.25 shows an example of a process containing "triggering information."

Task2 and task4 are handled by a resource. Task3 is time-dependent, and task1 requires an external trigger (for example, an EDI message). The only automatic task is task5.

The notion of triggering is of major importance. It is not the workflow system that is in charge, but the environment. The system cannot force a client to return a form; it cannot even force an employee to per-
form a work item at a particular time. It is easy to model the triggering mechanism in Petri net terms. To each transition belonging to a task requiring a trigger an extra input place is added. A token in such an extra input place represents the trigger. So a token appears in that extra input place when the trigger is recorded by the workflow system.

The triggering mechanism also shows that the *timing of an OR-split choice* is crucial. In figure 2.25, the timing of the nondeterministic choice between *task2* and *task3* is as late as possible. Once condition $c2$ has been met there are two possibilities. The first is that an employee begins the work item corresponding with *task2* before the moment specified for the performance of *task3* is reached. Alternatively, no employee takes the initiative to carry out *task2* before that moment. In the first case *task2* fires, in the second *task3* fires. A choice between the two alternatives thus is delayed until the moment when the first trigger is received. Because it is not known in advance which one will be activated, the implicit OR-split in the form of place $c2$ cannot be replaced by an explicit OR-split in the form of one or two additional transitions. So the OR-split comes in two forms: *implicit* and *explicit*. Figure 2.26 shows these diagrammatically.

Like the firing of a transition, an activity—that is, the actual performance of a task for a specific case—is an atomic unit. It thus is always carried out in full. However, a fault may occur during the performance of the task related to the activity. For example, it may make use of a resource (such as an employee) which interrupts it for some reason or another. An employee may notice, say, that certain data required to carry out the task are missing. Or the activity may use an application (such as a program for calculating interest charges) that crashes while performing

![Implicit OR-split](image1)
![Explicit OR-split](image2)

*Figure 2.26*

There is an essential difference between the implicit and explicit OR-split

- Implicit OR-split
- Explicit OR-split
the task. Moreover a failure in the workflow system itself—perhaps due to a system error—during an activity cannot be ruled out.

In all such cases, a so-called rollback is required. This involves returning the workflow system to its state prior to the start of the activity. Following the rollback, the activity can be restarted. Only when the activity has been successfully completed does a so-called commit occur and all changes made become definitive. As far as the process is concerned, a rollback is very simple: the case attributes and all valid conditions are returned to their original values. For the application (which has been cut off in the middle of performing a task), a rollback can be more complicated.

2.3.4 Example: Travel agency

Let us consider an example where triggers play an important role. To organize a trip, a travel agency executes several tasks. First the customer is registered. Then an employee searches for opportunities which are communicated to the customer. Then the customer will be contacted to find out whether she or he is still interested in the trip of this agency and whether more alternatives are desired. There are three possibilities: (1) the customer is not interested at all, (2) the customer would like to see more alternatives, and (3) the customer selects an opportunity. If the customer selects a trip, the trip is booked. In parallel, one or two types of insurance are prepared if they are desired. A customer can take insurance for trip cancellation or/and for baggage loss. Note that a customer can decide not to take any insurance, just trip cancellation insurance, just Baggage loss insurance, or both types of insurance. Two weeks before the start date of the trip the documents are sent to the customer. A trip can be cancelled at any time after completing the booking process (including the insurance) and before the start date. Note that customers who are not insured for trip cancellation can cancel the trip (but will get no refund).

Based on this informal description, we create the corresponding process using the constructs introduced in this chapter. Figure 2.27 shows the result.

The process, like any workflow process in this book, has a source place which serves as the start condition (i.e., case creation) and a sink place which serves as the end condition (i.e., case completion). First, the tasks
Figure 2.27
The travel agency

register, search, communicate, and contact_cust are executed sequentially. Task contact_cust is an OR-split with three possible outcomes: (1) the customer is not interested at all, that is, a token is put into end, (2) the customer would like to see more alternatives, that is, a token is put into c2, and (3) the customer selects an opportunity, that is, a token is put into c15 to initiate the booking. Tasks AND_split and AND_join have just been added for routing purposes. These routing tasks enable the parallel execution of the booking and insurance tasks. The task book corresponds to the actual booking of the trip. Tasks insurancel and insurance2 correspond to handling both types of insurance. Since both types of insurance are optional, there is a bypass for each of these tasks. The OR-split insurancel ? allows for a bypass of task insurancel by putting a token in c11. After handling the booking and optional insurances the AND-join puts a token in c13. The remainder of the process is, from the viewpoint of triggers, very interesting. Note that all tasks executed before this point are either tasks that require a resource trigger or automatic tasks added for routing purposes only. The downward-facing arrows denote the resource triggers. If the case is in c13, then the normal flow of execution is to first execute task send_documents and then execute start_trip. Note that task send_documents requires both a resource
trigger and a time trigger. These two triggers indicate that two weeks before the beginning of the trip a worker sends the documents to the customer. Task \textit{start\_trip} has been added for routing purposes and requires a time trigger. Without task \textit{start\_trip}, that is, putting the token in \textit{end} after sending the documents, it would have been impossible to cancel the trip after sending the documents. Task \textit{cancel} is an explicit OR-join and requires both a resource trigger and an external trigger. This task is only executed if it is triggered by the customer. Task \textit{cancel} can only be executed when the case is in \textit{c13} or \textit{c14}, that is, after handling the booking and insurance related tasks and before the trip starts.

Using the travel agency example, we point out two guidelines for modeling. The first guideline concerns the use of OR-joins. OR-join tasks should be avoided as much as possible. In most situations it is possible to use places/conditions instead of explicitly modeling OR-join tasks. If an OR-join task has two or more input conditions and these conditions are not input for any other task, then these conditions can be fused together because, from a semantical point of view, they are identical. As a result the number of elements in the diagram is reduced and there is no need to use an OR-join. For example, place \textit{c2} in figure 2.27 can be split into two conditions; one condition for new cases and one condition for cases that require more work. Such a split would introduce the need for an OR-join task \textit{search}. The resulting diagram only becomes more complex without changing the actual behavior. Therefore we prefer the solution with one condition \textit{c2} with two incoming arcs. Only in rare situations are OR-join tasks needed to obtain the desired behavior. Consider for example figure 2.27. Task \textit{cancel} is an OR-join. It is not possible to remove this OR-join by fusing the input conditions \textit{c13} and \textit{c14}. Conditions \textit{c13} and \textit{c14} correspond to different states, that is, in \textit{c13} \textit{send\_documents} is enabled and in \textit{c14} \textit{start\_trip} is enabled. The second guideline for modeling concerns the use of triggers for the first task in the process. In figure 2.27 we could have added an external trigger to task \textit{register}. This trigger would correspond to the request of the customer. Another interpretation is that the request of the customer corresponds to the creation of the initial token in condition \textit{start}. This interpretation is used in figure 2.27. Therefore the external trigger was not added to task \textit{register}. In this book we prefer to use this interpretation. However the interpretation that the
first task requires an external trigger to initiate the process is also allowed.

And finally ... In this chapter, we have introduced a process-modeling technique for the specification of workflows. It is based upon the theory of Petri nets and has a number of advantages. First, the technique is graphical and easy to apply. As we have seen using several examples, workflow concepts can be illustrated elegantly using Petri nets. Second, it is a technique with a good formal foundation: the meaning of each process is precisely defined. As a result, we have for example discovered that two types of OR-split exist. Another important advantage over many other process-modeling techniques is the fact that (interim) states are explicitly indicated. This enables us to differentiate between an implicit and an explicit OR-split. Explicit states also make it conceptually easier to cancel cases. Cancellation can be achieved simply by removing all the tokens belonging to that case. An explicit notion of states is also essential when transferring a case from one workflow system to another. Finally—because Petri nets have a formal basis—various analytical methods are possible.

EXERCISES

Exercises Classical Petri Nets

Exercise 2.1    German traffic light
There are some differences between traffic lights in different countries. The traffic lights described in this chapter are Dutch traffic lights. The traffic lights in Germany have an extra phase in their cycle. German traffic lights do not turn suddenly from red to green, but rather give an additional yellow light just before turning to green.

(a) Identify the possible states and model the transition system. A transition system lists all possible states and state transitions.
(b) Provide a Petri net that is able to behave like a German traffic light. There should be three places indicating the state of each light and all state transitions of the transition system should be supported.
(c) Give a Petri net that exactly behaves like a German traffic light. Make sure that the Petri net does not allow state transitions that are not possible.
Exercise 2.2  Project X
A secret project by the government (let's call it Project X) will be executed by one person and consists of 6 tasks: A, B, C, D, E, and F. Figure 2.28 specifies the order in which the tasks need to be executed (precedence graph, cf. PERT/CPM). A possible execution trace is for example ABDCEF.

(a) Model the project in terms of a classical Petri net.
(b) How does one model so that E is optional?
(c) How does one model so that D and E should be executed consecutively, that is, B and C are not allowed between D and E?

Exercise 2.3  Railnet
A circular rail network consists of four tracks. Each track is in one of the following three states:

- Busy, that is, there is a train on the track.
- Claimed, that is, a train has successfully requested access to the track.
- Free, that is, neither busy nor claimed.

There are two trains driving on the circular track. The track where a train resides is busy. To move to the next track a train first claims the next track. Only free tracks can be claimed. Busy tracks are released the moment the train moves to another track. One can abstract from the identity of trains only the state of the rail network is considered.

(a) Model the dynamic behavior of the rail network in terms of a Petri net.
(b) Is it easy to model the situation with 10 tracks (160 states)?
Exercise 2.4    Binary counter
The following (binary) counter is to be modeled as a Petri net. The marking of a place represents a binary value (1 or 0). The combination of the markings of these places represents the natural number that is displayed by the counter. For example, the binary number 101, that is, 5, marks two places corresponding to a "1" (i.e., the places $2^2$ and $2^0$) and one place corresponding to a "0" (i.e., the place $2^1$). Make a model of a counter able to count from 0 to 7.

Exercises High-Level Petri Nets

Exercise 2.5    Driving school
A driving school is trying to set up an information system to track the progress of the students' training and the deployment of instructors. As a starting point for a formal process model the following description can be used.

New students register with the driving school. A registered student takes one or more driving lessons followed by an examination. Each driving lesson has a beginning and an end. Instructors give driving lessons. The driving school has five instructors. Each driving lesson is followed by either another lesson or an examination. The examination has a beginning and an end and is supervised by a driving examiner. In total there are ten driving examiners. For the outcome of an examination there are three possibilities:

1. The student passes and leaves the driving school.
2. The student fails and takes additional lessons in order to try again.
3. The student fails and gives up.

(a) Model the driving school in terms of a classical Petri net.
(b) Use a colored Petri net to model that one takes ten lessons before taking the exam and people will drop out if they fail three times.
(c) Add time to model that a lesson takes one hour and an exam thirty minutes.

Exercise 2.6    Bicycle factory
A factory produces bicycles (just one type). The Bill Of Materials (BOM) is given in figure 2.29.
Suppliers deliver the raw materials. First the frame and two pedals are assembled. This takes twenty minutes and is done by a machine of type B. The other two assembly steps are defined in a similar fashion (see figure 2.29). Finally, the end product is delivered after three assembly steps. The factory has three machines of type A, and seven machines of type B. Each of the machines has a capacity 1, that is, a machine is either free or busy.

(a) Model the factory in terms of a Petri net. Make sure to model the states of the machines (busy/free) explicitly and abstract from time.

(b) Add time to model the temporal behavior. What is the maximal throughput per hour?

Workflow Process Definitions

Exercise 2.7  Insurance company
Insurance company X processes claims that result from traffic accidents with cars where customers of X are involved in. Therefore, it uses the following procedure for the processing of the insurance claims.
Chapter 2

Every claim, reported by a customer, is registered by an employee of department CD (where CD is Car Damages). After the registration of the claim, the insurance claim is classified by a claim handler of rank A or B within CD. There are two categories: simple and complex claims. For simple claims two tasks need to be executed: check insurance and phone garage. These tasks are independent of each other. The complex claims require three tasks to be executed: check insurance, check damage history, and phone garage. These tasks need to be executed sequentially in the order specified. Both for the simple and complex claims, the tasks are done by employees of department CD. After executing the two respectively three tasks a decision is made. This decision is made by a claim handler of rank A and has two possible outcomes: OK (positive) or NOK (negative). If the decision is positive, then insurance company X will pay. An employee of the finance department handles the payment. In any event, the insurance company sends a letter to the customer who sent the claim. An employee of the department CD writes this letter.

Model the workflow by making a process definition in terms of a Petri net using the techniques introduced in this chapter.

Exercise 2.8 Complaints handling
Each year travel agency Y has to process a lot of complaints (about 10,000). There is a special department for the processing of complaints (department C). There is also an internal department called logistics (department L) which takes care of the registration of incoming complaints and the archiving of processed complaints. The following procedure is used to handle these complaints.

An employee of department L first registers every incoming complaint. After registration a form is sent to the customer with questions about the nature of the complaint. This is done by an employee of department C. There are two possibilities: the customer returns the form within two weeks or he does not. If the form is returned, it is processed automatically resulting in a report that can be used for the actual processing of the complaint. If the form is not returned on time, a time-out occurs resulting in an empty report. Note that this does not necessarily mean that the complaint is discarded. After registration, that is, in parallel with the form handling, the preparation for the actual processing is started.
First, the complaint is evaluated by a complaint manager of department C. Evaluation shows that either further processing is needed or it is not. Note that this decision does not depend on the form handling. If no further processing is required and the form is handled, the complaint is archived. If further processing is required, an employee of the complaints department executes the task "process complaint" (this is the actual processing where certain actions are proposed if needed). For the actual processing of the complaint, the report resulting from the form handling is used. Note that the report can be empty. The result of task "process complaint" is checked by a complaint manager. If the result is not OK, task "process complaint" is executed again. This is repeated until the result is acceptable. If the result is accepted, an employee of the department C executes the proposed actions. After this the processed complaint is archived by an employee of department L.

Give the process, that is, model the workflow by making a process definition in terms of a Petri net.

Exercise 2.9 Let's have a party
A group of students wants to set up an agency to organize parties. The customer should indicate the amount of money to be spent, the number of persons the party is meant for, and the area in which the party is to be given. With that information, the agency looks for a suitable location and takes care of the rest.

Locations are indoors or outdoors. If the location is indoors, a room is to be hired. In case of an outdoor location, however, a party tent and a terrain have to be arranged, possibly along with a permit for making noise (music). There are two sorts of music: live or CDs. The choice between these alternatives is not made by the customer, but by the agency itself: live music is preferred, but expensive, so most parties will have to do with CDs. CDs are also chosen if there is not enough time left to ask a band. If CDs are chosen, a sound system has to be arranged. In case of live music, however, things are more complicated. First, a band is selected. Then this band is sent a letter inviting it to play on this party. If the band does not react within a week, a new band is selected and the procedure is repeated. If they do react, there are again two possibilities: they are interested or not interested. In the latter case, a new band is
selected and the procedure is repeated. In the first case, however, the band is not hired immediately. First the agency should see and hear the band to see if they are good enough. Because the students only take the best, about thirty percent of the bands is considered good enough. For the other seventy percent, a new band is selected, and so on. If the students cannot find a band quickly enough, they switch to CDs. Of course, the bands that have been hired before do not have to be evaluated first. They're hired immediately. After taking care of the location and the music, they also take care of food and drinks. In case of a band, they order extra food and drinks for the musicians. To make sure everything is fine, the students take a look at the party when it is being held. After that, a bill is sent to the customer.

(a) Model the workflow by making a process definition in terms of a Petri net using the techniques introduced in this chapter. Assign triggers to tasks whenever appropriate.

(b) Analyze the process and investigate possible improvements.
3  
Management of Workflows

3.1  Resource Management Concepts

Using the definition of a process, we can indicate *which* tasks need to be performed for a particular category of case. We can also show the *order* in which they must be carried out. However, the process definition does not indicate *who* should do it. But the way in which the work items are allocated to resources (people and/or machines) is very important to the efficiency and effectiveness of the workflow. In this chapter, we shall concentrate upon the management of resources and the link between a process definition and the resources available. We shall also pay attention to improving workflows.

3.1.1  The resource

A workflow system focuses upon supporting a business process. In this process, work is carried out by means of production, also called *resources*. In an administrative environment, the term resource primarily refers to office staff. However, a doctor, a printer, a doorman, and an assembly robot are all examples of resources. The basic characteristic of a resource is that it is able to carry out particular tasks. We also assume that each resource is uniquely identifiable and has a certain capacity. In this chapter, we shall confine ourselves to resources with a capacity of one. In other words, each resource may be working on no more than one activity at any given time. This does not, however, have to be the case in practice.

3.1.2  Resource classification

In general, a resource is permitted to carry out a limited number of tasks. In a bank, for example, a teller is not allowed to grant a mortgage. A task
usually can be performed only by a limited number of resources. Because it is impracticable to indicate which resources are able to carry out each task, we classify them using resource classes. A resource class is a group of resources. For example, the resource class Counter_Staff may consist of the people Annie, Hank, Mandy, Jack, and Tom. A resource may belong to more than one class. So Annie, say, could be a member of both the Counter_Staff and the Travel_Agent categories. In general, we differentiate between two forms of resource classification: (1) that based upon functional properties and (2) that based upon position within the organization.

A functionally based resource class is known as a role. It is also referred to as a function or qualification. A role is a group of resources, each of which has a number of specific skills. Such resource classes as Counter_Staff, Travel_Agent, Assessor, C_Executive, Administrator, Printer, Hospital'_Bed, and Junior_Doctor are examples of roles. By linking a task to the correct role, one can ensure that the resource carrying out the task is sufficiently qualified (and authorized).

Resources can also be classified according to their place in the organization. Under this definition fall such resource classes as Sales_Department, Purchasing_Department, Team_2, and Atlanta_Branch. A resource class based upon organizational rather than functional characteristics also is called an organizational unit. This form of classification can be used to ensure that a task is carried out at the right place in the organization.

Figure 3.1 shows a resource classification diagrammatically. In total, there are eight resource classes. Of these, the resource classes Atlanta, Denver, Purchasing_Department, and Sales_Department are examples of organizational units. So the resource Jack works at the Atlanta branch in the Sales^Department. The remaining resource classes are based upon functional characteristics. The resource class Secretary, for example, contains all those resources which are qualified to act as a secretary. As we can see in figure 3.1, resource classes may overlap. It is even possible for one resource class to be a subset of another, larger one. The resource class Salesperson, for example, is contained entirely within the resource class Office_Staff. We can use a classification similar to the one shown in figure 3.1 to link a particular task to the appropriate resource(s). Say we need a salesperson based in Denver. In this case, only one resource
qualifies: Frank. If we need a secretary in the Sales_Department, two resources are possible: Mary and Carl.

As already indicated, in most cases a resource classification consists of two parts. We call that part containing the functional structure the role model and that containing the organizational units the organization chart. Note that the term organization chart usually has a broader meaning, referring to the hierarchical structure of the organization.

### 3.1.3 Allocating activities to resources

In order to ensure that each activity is performed by a suitable resource, we provide each task in the process definition with an allocation principle (see figure 3.2). This specifies which preconditions the resource must meet. In most cases, the allocation specifies both a role and an organizational unit. The resource then must belong to the intersection of these two resource classes. However it is also possible to define a much more complex allocation. From figure 3.1, for example, we could specify the resource classes Office_Staff and Atlanta, but exclude Salesperson. The task with this allocation rule therefore may be carried out only by an office worker in Atlanta who is not a salesperson. The allocation may also depend upon the attributes of the case for which the task must be
Figure 3.2
Allocation principles link the process definition with the resource classification

carried out. Depending upon these attributes we can, for example, select the organizational unit. To assess an insurance claim, for instance, we would select the nearest branch of the company. In such a case, we should use the customer's address as a case attribute. When the Internal Revenue Service deals with a tax return, the allocation may depend upon the name of the person making the return. A particular assessment team is selected based upon the name. In this case, it is of course the person's name that acts as a case attribute.

By making careful use of the case attributes, we can also ensure that an activity is performed by a specific resource. But the opposite is also possible. In a bank, for example, it may be that one member of staff is not allowed to perform two successive tasks on the same case. We call this separation of function. This term is taken from accountancy. Here, it is important that certain tasks not be carried out by the same person in order to prevent fraud. The financial settlement of a travel-expenses claim, for example, should not be done by the person who authorized the journey. The objective of separation of function is to combat abuse. Because each case is dealt with by several people, it becomes more difficult to commit fraud. If a number of successive tasks do need to be carried out by, or under the authority of, a single employee, then that person is referred to as a case manager. Because she is largely responsible for a case, she is naturally more involved in it. The appointment of a manager for each case can result in a better service to the customer and more rapid completion because of greater familiarity with the work.

By providing a task with an allocation principle, we specify the preconditions that the resource must meet. In most cases, there is more than
one resource that may carry out the activity associated with a particular work item.

At the heart of a workflow system is the workflow engine. This ensures the actual enactment of a specified workflow. One of its core tasks is to allocate work items to resources. In doing so, it must take into account the resource classes specified, as well as such things as separation of function and case management. In many cases, the workflow engine nevertheless is able to choose between a number of resources when allocating work. It then has to decide which resource will carry out the activity. We shall return to this later.

3.2 Resource Management in More Detail

The allocation of resources to activities is not a simple issue. As we have seen, such concepts as the task, the case, the work item, the activity, the case attributes, the resource, the resource class, the role, the organizational unit, and allocation are all closely connected with one another. For the sake of clarity, we therefore make use of a simple data model which summarizes the concepts and their mutual relationships. Figure 3.3 shows an entity relationship (ER) diagram. Broadly speaking, this diagram consists of two types of elements: entity types and relationship types. The former is indicated using a rectangle and represents a group of entities. For example, the entity type task contains all the tasks that form part of a process. Relationship types are illustrated using a diamond. This represents a group of relationships. So the relationship type belongs_to, for example, contains a collection of relationships between resources and resource classes. If there exists a relationship between resource r and a resource class c, then this means that r belongs to c.

The relationship type of between task and work item indicates to which task a work item relates. Each work item has a relationship with precisely one task and each task may have an arbitrary number of work items (say N) associated to it. This is shown using the symbols 1 and N. These therefore refer to the cardinality of the relationship of. We can also say that there exists a 1-on-N relationship. In other words, each work item relates to precisely one case. It may be possible for more than one work item to have a relationship with the same case. This may, for example, result from parallel routing.
An entity of the entity type *activity* relates to the actual performance of a work item. So, like a work item, an activity relates to a single case and a single task. Moreover, zero or one resources are also attached to each activity. The relationship type *belongs_to* is an example of an M-to-N relationship, which specifies that a resource may belong to several resource classes and a resource class may contain several resources. A role and an organizational unit are examples of resource classes. Hence the entity types *role* and *organizational unit* are associated with the entity type *resource class* through a so-called ISA relationship type. This indicates that roles and organizational units are special cases of resource classes.

In the ER diagram, we differentiate between a specific case and a case type. The latter corresponds with a process: it is the category of cases that...
Management of Workflows

can be dealt with by that process. The ER diagram also indicates that there exists a one-on-one relationship between the case type and the process. We also differentiate between case attributes and specific case attributes associated with a specific case. The former refers to a logical name that expresses a particular property, the latter to the value of an attribute in a specific case that is in progress. The entity type allocation determines which conditions the relationship type by between the entity types activity and resource must fulfill.

As noted earlier, the preconditions formulated in the allocation policy can become highly complex. After all, an allocation relates tasks, resource classes, case attributes, and resources to each other. Each task has one or more allocations. And an allocation may depend upon one or more case attributes. In most cases, an allocation will point to the intersection of a role and an organizational unit. In special cases, though, a specific resource may be excluded (separation of function) or selected (case manager).

The ER diagram can only provide an impression of the static aspects of resource management. We can regard such a diagram as a "snapshot" of resource management at a particular moment, that is, the diagram only describes the structure of all possible states. Its dynamic aspects are not shown in figure 3.3. To illustrate these, we must look at the process shown in figure 3.4.

The process handle complaint consists of eight tasks, of which three are automatically handled (they do not involve intervention by a resource). Moreover there are four resource classes. Two of these are based upon functional characteristics: Employee and Assessor. Alongside these two roles there are two further resource classes based upon organizational characteristics: Complaints and Finance. These correspond with two of the company departments. Figure 3.4 also shows diagrammatically the allocation for each task. The task contact_dient is linked with the role Employee and the organizational unit Complaints. This means that an employee in the complaints department is needed to approach the client. A resource from the intersection of the resource classes Employee and Complaints also is required for the tasks contact_department and send_letter. For the task pay, an employee from the financial department is needed. The task assess is carried out by a resource from the intersection of the resource classes Assessor and Complaints. In figure 3.5, these
Figure 3.4
The process "handle complaint" and the resource classes involved in it

<table>
<thead>
<tr>
<th>Resource class</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>John, Jim, Liz, Jack, Mandy, Carl</td>
</tr>
<tr>
<td>Assessor</td>
<td>Mandy, Carl, John, Jim, Mandy, Carl</td>
</tr>
<tr>
<td>Complaints</td>
<td>John, Jim, Mandy, Carl</td>
</tr>
<tr>
<td>Finances</td>
<td>Liz, Jack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
<th>Organizational unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
<tr>
<td>contact_client</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
<tr>
<td>contact_dept.</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
<tr>
<td>collect</td>
<td>Assessor</td>
<td>Complaints</td>
</tr>
<tr>
<td>assess</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
<tr>
<td>pay</td>
<td>Employee</td>
<td>Finances</td>
</tr>
<tr>
<td>send_letter</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
<tr>
<td>file</td>
<td>Employee</td>
<td>Complaints</td>
</tr>
</tbody>
</table>

Figure 3.5
A summary of the composition of each resource class and those required for each case
In the state illustrated, there are six complaints in progress allocations are shown again, but in table form. The composition of each resource class is also given.

In figure 3.5 we see, for example, that Mandy belongs to the resource classes Employee, Assessor, and Complaints. She thus can carry out any task except pay. Liz and Jack, on the other hand, can only carry out the task pay.

Figure 3.6 shows the states of six cases. Case 1 has been assessed positively, resulting in a work item (pay). (In other words, the task pay is enabled for case 1.) For case 2, the activity assess is being performed. Based upon the states shown in figure 3.6, we can establish the relevant work items and activities. These are shown in the table in figure 3.7. However the opposite is not possible. Based upon the table in figure 3.7, we cannot directly work out the state of each case. For example, it is impossible to tell directly from the table that there is a token in the place corresponding to condition c3.

There is a total of four work items. Each corresponds with the potential performance of a task for a particular case. Note that in the situation depicted in figure 3.6 there are two work items for case 5. This is because of parallel routing, which enables the tasks contact_client and contact_department simultaneously. There are three activities. Each of these corresponds with the actual performance of a task for a particular case. The first corresponds with the performance of the task assess for case 2 by resource Mandy. The second is carried out by Jim: the task contact_
# Chapter 3

## Work items

<table>
<thead>
<tr>
<th>Case</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case1</td>
<td>pay</td>
</tr>
<tr>
<td>Case3</td>
<td>assess</td>
</tr>
<tr>
<td>Case5</td>
<td>contact_client</td>
</tr>
<tr>
<td>Case5</td>
<td>contact_dept.</td>
</tr>
</tbody>
</table>

## Activities

<table>
<thead>
<tr>
<th>Case</th>
<th>Task</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case2</td>
<td>assess</td>
<td>Mandy</td>
</tr>
<tr>
<td>Case4</td>
<td>contact_dept.</td>
<td>Jim</td>
</tr>
<tr>
<td>Case6</td>
<td>record</td>
<td>-</td>
</tr>
</tbody>
</table>

**Figure 3.7**
The work items and activities for the state illustrated in figure 3.6

*department* for case 4. The last is the task *record* for case 6. As shown in figure 3.5, no resource is required for this.

Each of the work items shown in figure 3.7 can, in principle, be transformed into an activity. The first (task *pay* for case 1) requires a resource from the intersection of the resource classes *Employee* and *Finances*. Both Liz and Jack thus qualify. The second (task *assess* for case 3) can only be carried out by a resource from the intersection of *Assessor* and *Complaints*. Since Mandy is already busy assessing case 2, Carl is the only resource able to perform this work item immediately. The other two work items require a resource from the intersection of *Employee* and *Complaints*.

### 3.2.1 Allocation principles

The objective of a workflow system is to complete work items as quickly as possible. After all, a hold up affecting work items can result in the case as a whole taking longer. In order to transform work items into activities, two decisions always need to be made:

- **In what order are the work items transformed into activities?** If there exists an excess of work items at particular times, we cannot transform each into an activity immediately. There may, after all, be more work items than there are resources available. If this is the case, then a choice must be made as to the order in which the work items are selected.
- **By which resource are the activities carried out?** Because not all resources are the same, it may matter to which resource a particular work item is allocated. A specialist resource, for example, can carry out certain tasks more quickly. It may also be sensible to keep a flexible resource—one that is a member of a large number of resource classes—free for as long as possible.
It goes without saying that these two decisions are closely interrelated. The order can be important when selecting a resource. Conversely the choice of a resource can affect the order in which work items are transformed into activities.

Many different heuristics can be applied to select a particular order. In particular, we can borrow the various queueing disciplines for production management that are used in factories. The routing of a case through several resources exhibits many similarities with the routing of a product through machines in a production department. Some common queueing disciplines are as follows:

- **First-In, First-Out (FIFO).** If work items are dealt with in the order in which they are created, we refer to a FIFO discipline. Rather than the time when the work item was generated, we can also use the moment when the case as a whole was created. FIFO queueing is a simple and robust allocation rule and is the most widely used in practice.
- **Last-In, First-Out (LIFO).** LIFO is the opposite of FIFO. In this arrangement, the work items created most recently are dealt with first. In certain cases, this (unfair) allocation rule can lead to a higher average level of service.
- **Shortest Processing Time (SPT).** We can sometimes estimate in advance, using the attributes of a case, how much time is required to perform an activity. A distinction can often be made between easy and difficult cases, and between simple and time-consuming tasks. By selecting first those work items that take the least time, it is often possible to reduce the average flow time of cases. It is also possible, however, to imagine situations in which it is actually better to give time-consuming tasks priority over simplest ones. We then refer to a Longest Processing Time (LPT) queueing discipline.
- **Shortest Rest-Processing Time (SRPT).** If we have some insight into the time required to carry out particular activities for a given case, and into the routing of that case, then we can estimate its remaining total net processing time. By always prioritizing the case with the shortest remaining processing time, the quantity of work in progress (WIP) is generally minimized. If, conversely, we select the case with the longest remaining processing time, then we refer to a Longest Rest-Processing Time (LRPT) queueing discipline.
- **Earliest Due Date (EDD).** An activity is always carried out in the context of a case. This was initiated at a certain time, and should preferably also be completed by a set time (the "due date"). The EDD queueing discipline determines the order based upon the case's deadline.
So a case that must be finished today takes priority over one that needs to be ready in a week. The tasks still to be carried out may also be taken into account when deciding the order.

Note that the information required by each queueing discipline can vary widely. FIFO needs virtually no information. SRPT, though, requires information about the expected processing times and the routing. There also exist very advanced queueing disciplines that take into account the work in progress, the expected supply of work, and the availability of resources. These disciplines are characterized by their use of the current state of the workflow or of forecasts of its future state.

When considering queueing disciplines, we thus far have assumed that the order is determined by the individual characteristics of a case. However it is also possible for it to be decided for a batch of cases. For a given batch, it is sometimes possible to improve the order using certain criteria.

In what order work items are transformed into activities is closely associated with the selection of the resource. If a work item could be carried out by more than one resource, then the following considerations come into play:

- **Let a resource practice its specialty.** A resource can often perform a large number of tasks. Usually, though, there are some in which it specializes. A tax inspector, for example, may be qualified to assess a whole range of tax returns but at the same time be specialized in those submitted by building contractors. It therefore is preferable to let this resource practice his specialty.

- **As far as possible, let a resource do similar tasks in succession.** Both people and machines require so-called set-up times. By this we mean the (additional) time required to begin performing a new task. The set-up time may, for example, be spent opening an application or getting used to a new task. By carrying out similar tasks one after the other, the set-up times can be cut down. Furthermore in the case of work of a repetitive nature, people can reduce their average processing time by using routine.

- **Strive for the greatest possible flexibility for the near future.** If we have a choice between two resources of equal value to perform a work item, it is wise to select the one that can carry fewer work items of other types. In other words, save the "generalists" until last. In the situation shown in figure 3.7, for example, it would not be sensible to allocate Carl to one of the work items for case 5. If we were to do so, all the resources from the resource class Assessor would be busy and case 3 could not proceed any further. By keeping the "generalists" free, flexibility for the near future is guaranteed.
So when allocating work items to resources, choices must continually be made. There are two ways in which this can be done:

- *The workflow engine matches work items and resources.* Within preset conditions, the workflow engine can choose which resource performs each work item. The resource itself thus is unable to choose. As soon as it has finished performing one activity, it is given a new work item. We refer to this as *push-driven:* the engine "pushes" work items onto resources.

- *The resources themselves match work items and resources.* In this scenario, it is the resources that take the initiative. Each has studied the work items that it is able to carry out. It then chooses one. We call this *pull-driven:* the resources "pull out" work items and all "eat" from the same basket of work items.

Usually an approach somewhere between push and pull-driven is taken. One common method is the pull principle supplemented by an ordering of the work items by the workflow engine. A resource thus sees an ordered list of the work items that it can carry out. This is supplied by the workflow engine, which sorts the work items according to such principles as FIFO, LIFO, SPT, or EDD. The resources preferably take the first work item on the list. They may, however—and for whatever reason—choose another. The advantage of this mixed approach is that the workflow engine is given an advisory role while the (human) resources still retain the freedom to decide what work they do.

### 3.3 Improving Workflows

A workflow system enables an organization to use and manage structured business processes. One important property of workflow systems is that, by comparison with classic information systems, it becomes easier to change business processes. Exchanging or combining tasks, or rearranging resource classes, are easy modifications. It therefore is interesting to examine how we can improve the workflows that are being managed by the system. Improvements influence performance criteria such as completion times, utilization of capacity, level of service, and flexibility.

#### 3.3.1 Bottlenecks in the workflow

When should the process, resource classification, or resource management be changed? If a workflow is not working properly, we can often
observe all types of symptoms. These can be compared with the functions of our body. Symptoms like headaches, diarrhea, nausea, or coughing indicate problems. In a workflow, there also are typical symptoms that betray the presence of a bottleneck that is obstructing its proper operation. Some typical symptoms are listed below:

- **Number of cases in progress (too) large.** If there are many cases in progress, this can indicate a problem. This large number can be caused by major fluctuations in the supply of cases or by a lack of flexibility in the resources. However, it may also be that the process contains too many steps that need to be passed through sequentially.

- **Completion time (too) long compared with actual processing time.** The actual processing time of a case sometimes forms only a small part of the total time it is in progress. If this is the case, there may be a whole range of possibilities for reducing the completion time.

- **Level of service (too) low.** A workflow's level of service is the degree to which the organization is able to complete cases within a certain deadline. If the completion time fluctuates widely, then there is low level of service. It is not possible to guarantee a particular completion time. A low level of service also exists when there are many "no sales" occurring. (By this, we mean the inability to take on potential cases due to the long waiting times.) When the client knows that it will take a long time to complete a case (say, a loan application), it will approach another company. A low level of service can indicate a lack of flexibility, a poorly designed process or a structural lack of capacity.

These three symptoms point to possible bottlenecks. To identify them we need benchmark values for these measures, for instance from comparable processes. Usually, it is not sensible to combat the symptoms using only emergency measures. It is important to tackle their causes.

To alert us to problems and to measure the performance of a particular workflow, we use performance indicators. These express the performance of a particular aspect of the workflow. In general, we distinguish between two groups of performance indicators:

- **External performance indicators (case-oriented).** The external performance indicators focus upon those aspects that are important to the environment of the workflow. For example, indicators of the average completion time and reliability of the completion time. Note that these indicators can be subdivided according to the specific properties of the case.
Management of Workflows

- **Internal performance indicators (resource-oriented)**. The internal performance indicators show what efforts are required to achieve the external performance (for example, the level of resource utilization, the number of cases per resource, the number of cases in progress, the number of rollbacks, and the rate of turnover). The latter is a measure of the speed at which cases proceed through the workflow system. It is calculated by dividing the length of a period (for example, a month) by the average completion time, or by dividing the average number of cases which come in during a period by the average number of cases in progress.

A poor external performance costs a lot of money. Consider, say, a bank: a long completion time for mortgage applications causes a loss of many clients. However, a good external performance can require a high degree of internal effort. Achieving a rapid completion time can, for example, require extra overtime or the allocation of additional resources. The objective of every organization is to minimize its total costs. As shown in figure 3.8, careful weighing of the costs of a poor external performance (no-sale costs) versus those of internal effort is required.

Nevertheless it is in many cases possible to improve the external performance of a workflow without allocating additional resources. Such an improvement can be achieved by restructuring the workflow or using a better allocation strategy.

---

*Figure 3.8*  
Weighing external performance versus internal effort
33.2  Business Process Re-engineering

Before focusing upon improving workflows, we shall consider the relationship between business process re-engineering (BPR) and workflow management. We can define BPR as the fundamental reconsideration of business processes. Its objective is to bring about entirely new business processes which enable drastic improvements to costs, quality, and service. In order to achieve this objective, radical changes are often necessary. For many administrative processes, the rise of workflow management systems is an "essential enabler" for BPR efforts. After all, the use of a workflow management system makes it easy to adapt processes.

The introduction of a workflow system makes it possible to work in a completely different way. Conversely, some BPR efforts result in the purchase of a workflow management system. Workflow management and BPR are natural partners. It is therefore important for work-process designers to be aware of the latest developments in BPR.

In their book *Re-engineering the Corporation*, Michael Hammer and James Champy write that BPR is characterized by four key words: fundamental, radical, dramatic, and process. The keyword *fundamental* indicates that, when revitalizing a business process, it is of great importance always to ask the elementary questions: why are we doing this, and why are we doing it like this? *Radical* means that the re-engineering must represent a complete break from the current way of working. BPR is not an improvement of the existing processes, but their replacement by completely new ones. The third keyword also refers to the fact that BPR must not effect merely marginal or superficial changes, but that these must be *dramatic* in terms of costs, service, and quality. But of all the keywords, *process* is perhaps the most important. In order to achieve a dramatic improvement, it is necessary to focus upon the business process. This means that the organization must be subordinated to the primary business process. To operate in a genuinely process-oriented way, one must abstract oneself from other aspects, such as people, functions, jobs, teams, and departments.

Process-oriented thinking is crucial in the use of workflow management systems. One of the great dangers threatening the successful introduction of a workflow system lies in simply computerizing existing (manual) practices. Supporting old processes with a workflow system will only deliver a limited amount of improvement. Dramatic improve-
ments are only possible if the old processes are separated from and replaced by new ones. One common error when introducing a workflow system is the unnecessary sequencing of tasks. The fact that a physical document can only be in one place at a time led to sequential routing in many old style processes. However, computerization of the document and the use of a workflow system enable parallel routing in many cases. It is important to structure the new process in such a way that parallel routing also becomes possible (see chapter 6).

3.3.3 Guidelines for (re)designing workflows

Inspired by many experiences in BPR, we are able to propose a number of rules of thumb (i.e., best practices) for the design or redesign of workflows. These relate to process design, resource classification, and the allocation of activities to tasks:

1. **First establish the objective of the process.** When designing a new workflow or changing an existing one, it is crucial to consider the role played by the process in the greater scheme of things. Why do we need the workflow at all? By reflecting upon this fundamental question, it is possible to define the new workflow without misleading presuppositions.

2. **Ignore the existence of resources when defining a process.** The process definition is independent of the potential offered by people and machines. If the allocation of work to resources is already being considered when drawing up the process definition, one runs the risk that the resulting process will not be the best one possible. First list which tasks are required and in what order they should be carried out. Only then link the tasks to resources. In other words, do not allow yourself to be distracted by the traditional structure of the organization when designing a process. In all, we recognize four phases in the (re)design of a workflow: (1) **What?**, (2) **Why?**, (3) **How?**, and (4) **Whom?**. Figure 3.9 shows these phases diagrammatically.

   During the first phase we select the process that needs to be redesigned. During the second we consider the objective of the process: what is its output, in terms of product delivered, and do we need this? During the third we determine the structure of the process. Only during the last phase do we focus upon allocating work to resources.

3. **As far as possible, make one person responsible for the processing of a case (case manager).** Processes supported by a workflow system can be quite complicated. For the client, it therefore is often very difficult to gauge the progress of a particular case. This is why it is sensible to appoint a manager for each case. He or she acts as a sort of buffer between
Select the workflow that has to be (re)designed.

First establish the objective of the workflow to be (re)designed.

Then establish the steps that must be carried out, and in what order.

Finally, establish the allocation of work to resources.

Figure 3.9
The four phases through which the (re)design of a workflow passes

the complicated process and the client. In doing so, it is important that the case manager behaves towards the client as if he or she is responsible for the entire process. This provides the client with a single point of contact, and the case manager feels more involved in the work. Note that the case manager is only responsible for the case itself. Other resources can be used to actually carry out the activities associated with the case.

4. **Check the need for each task.** Tasks are sometimes added for the sake of security: for example, monitoring tasks. Such tasks often are used as a stopgap to conceal a problem in one of the previous tasks. For the same reason, iterations should always be examined critically. In short, eliminate those tasks that add no value.

5. **Consider the scope of tasks.** A task is a logical unit of work. By combining separate tasks into one composite task, set-up times can be reduced. The involvement of the people performing them is also increased. However tasks should not be too large. Because a task always has to be performable in one go, without interruptions, "bite-size chunks" are desirable. Large tasks can also inhibit flexibility and make an advanced allocation of work impossible.
6. Strive for the simplest possible process. Complex process definitions lead to unmanageable processes. This is why it is important that a process not be unnecessarily complex. Processes can often be simplified by adding more "intelligence" to the tasks. If it is impossible to avoid a complex process, then it is essential to establish a clear hierarchical structure. When breaking down a process, it is important to ensure that tasks with a close relationship form part of the same subprocess. In addition, it is sensible to allow as few causal links as possible between different subprocesses. Ideally, each subprocess will have one entrance and one exit. However, the most critical consideration is that the process be understood by the people carrying out the work. If this is not the case, the result can be a difficult-to-manage process.

7. Carefully weigh a generic process versus several versions of the same process. Do not define a separate process for each type of case. Try to create a generic process that distinguishes between the various types of cases by using selective routing. Do not, though, attempt to handle two completely different types of cases in a single process. If a process begins with an OR-split which sends the case into a number of alternative subprocesses, then it is probably a good idea to use a number of separate subprocesses. Each of these will then correspond with a version of the same process.

8. Carefully weigh specialization versus generalization. The division of a generic task into two or more alternative tasks may have either a positive or a negative effect. One advantage can be that the tasks become better suited to the specific qualities of a resource. There can be drawbacks to specialization, though. It often detracts from the flexibility and accessibility of the process. It also can lead to monotonous work, which reduces motivation. Rather than specialization, the term triage is often used. This is the classification of cases in order to enable selective processing.

9. As far possible, try to achieve parallel processing of tasks. Always Consider whether tasks can be performed in parallel. If two tasks can be carried out independently of one another, then it is very important that the process allows for their parallel execution. The unnecessary introduction of sequential order relationships results in longer completion times and the inefficient use of resources.

10. Investigate the new opportunities opened up by recent developments in networking and (distributed) databases. The elimination of physical barriers resulting from such developments as the computerization of documents often makes possible entirely new process structures. Tasks that previously had to be performed in sequence can be carried out in parallel following the introduction of, say, a workflow package.
11. *Treat geographically scattered resources as if they are centralized.* The introduction of a workflow system lowers the physical barriers between the various sections of an organization. It makes it easier for two organizational units to exchange work. If team A is struggling with a flood of work, but team B is operating below capacity, it is logical to transfer work from A to B. It is even better to treat geographically scattered resources as if they are centralized. This enables resources to be allocated to those places where most of the work is waiting.

12. *Allow a resource to practice its specialty.* As mentioned earlier, it is important to make use of a resource's specific qualities.

13. *As far as possible, allow a resource to perform similar tasks in succession.* By performing similar tasks one after the other, set-up times can be reduced and the benefits of routine working can be exploited.

14. *Try to achieve as much flexibility as possible for the near future.* When allocating work to resources, it is sensible to retain as much flexibility for the near future as possible.

15. *Allow a resource to work as much as possible on the same case.* If an employee performs a number of successive tasks for a specific case, the total processing time is usually shorter than if different employees carry out those tasks. Less time is taken because the member of staff does not have to "get used" to each new case.

Based upon the guidelines listed above, workflows can be designed that result in the efficient and effective processing of cases. A number of these guidelines highlight the fact that a balance needs to be struck between two or more alternatives. In many cases, which should be chosen can only be decided following a thorough analysis. Such an analysis is usually of quantitative aspects, with the emphasis being placed upon such performance indicators as average completion time, level of service, and utilization of capacity. There are various analytical techniques available for establishing these performance indicators using a modeled workflow. A number of these are addressed in the next chapter.

**EXERCISES**

Exercise 3.1  Insurance company
Consider the insurance company described in exercise 2.7.

(a) Make a resource classification with relations between roles (qualifications) and groups (organizational units).

(b) Assign a role and a group to each task in the process model.
Exercise 3.2  Complaints handling
Consider the complaints handling process described in exercise 2.8.
fa) Make a resource classification with relations between roles (qualifications) and groups (organizational units), (b) Assign a role and a group to each task in the process model.

Exercise 3.3  Employment Office
Agency "Job Shop" accepts requests for new employees by companies all over the country. Requests can be sent by e-mail, by mail, or by phone to one of the agencies in Eindhoven and Leeuwarden. Handling these requests is a job for someone in the department of business relations (BR). For the Eindhoven agency this job is done by Johan in Leeuwarden Sietse, who is responsible for BR. The first thing being done is sending an acknowledgement back to indicate that the request has been received. Then "Job Shop" has several options: they always look in their database to find suitable people, but they can also place an advertisement in some of the greater papers in the country to ask for people, as well. Placing an ad is a job for those in public relations (PR): Jaap and Anke in Eindhoven, Rinske in Leeuwarden. The manager decides whether or not this option should be used. Being a manager is a job fulfilled by Ahmed (Eindhoven) and Dion (Leeuwarden).

The actual searching in the database is done by someone in recruitment. All candidates for the job get a marking that will be used later.

People who react to the ad can do this by phone, by completing a form (found at Internet), or by dropping off a letter with their data at the office. Someone from recruitment processes the data in the form/letter by adding it to the database and by marking candidates for the job. If someone uses the phone, a member from recruitment will interview this person to get his/her data for the database. Again, a marking is placed if the person fits the requirements for the job.

The Eindhoven recruitment team is formed by Annelies, Manja, and the people of both PR and BR. In Leeuwarden, Anja, Hakan, Rinske (also PR), and Sietse (also BR) take care of new people.

After some time, the deadline for a job expires and a candidate has to be chosen from the ones marked in the database. Reactions to the ad, if placed, will not be processed anymore from then on. One by one, the candidates will be called by someone in the recruitment team until
someone has been found. In this call, she gets an invitation to come to the office to discuss the possible new job. Of course people may refuse to come. However, if someone agrees to come to the office, an appointment is made and she gets an interview with one of the employees (recruitment) of "Job Shop." Immediately after this interview an evaluation is made and the candidate is told whether or not she will be chosen. If no candidate can be found, or when no one is suitable for the job, a letter is sent to the company. 

Once someone has been chosen, she gets a letter with all the data needed to prepare for the new job. This letter is composed by someone from recruitment. Also a letter is sent by BR to the company for which the new employee has been found. In this, all relevant data concerning the new employee is listed. Of course, the database will have to be updated in order to reflect the new status of this person. This is done after sending the letters, by the same person from recruitment who sent the letter.

Maintenance of the database in both agencies is done by Mahroud, the IT specialist.

(a) Make a resource classification with relations between roles (qualifications) and groups (organizational units).
(b) Construct a process model of the process sketched above.

Exercise 3.4    Have a nice flight with CRASH

We will look at the preparation of a flight plan for the aircraft of the company "CRASH" (Cheap and Reliable Aerial Shipments). This company transports freight for customers from place Y to place Z.

Each customer sends a form describing the freight and the wishes she has about it. Upon receipt of such a form, a secretary makes a copy of it. The original is taken to a loadmaster, who, with his knowledge of the capacity of all the company's aircraft, will decide which aircraft will be used. The copy is sent to the navigator. The navigator, responsible for setting out the flight plan, takes a flight plan paper and fills in the date, her data (name and employee number) and the client number. Then the navigator has to check the following things in sequence before planning the flight:

• What freight will be taken and, more important, where does it have to be delivered? Together with the loadmaster this will be discussed. The
type of aircraft and its payload will influence the flight path: perhaps some extra stops are needed to refuel.

- What are the weather conditions? For this the navigator goes to the north side of the company's building to meet with someone in meteorology. Together they will discuss the weather for that day and that person will put the information on a map.

- There might be some exceptions: some areas have to be avoided because of military exercises, etc. At the south side of the building, the directors have their room. They know all about those exceptions and will tell the navigator what she needs to know. The same map is used to draw the areas for which exceptions hold.

Once the navigator has gathered these three pieces of information, she can start planning the flight in her room at the west side. For this she uses a special form, not the form she already has filled out in part. The reason for this is that she wants to be able to make corrections without spoiling the official flight plan. After that, she takes the flight plan to the directors. One of them will check this flight plan with other, already approved flight plans. This will ensure that collisions with other aircraft because of incorrect flight plans will be prevented. Also some mistakes the navigator might have made, however small the chances of that are, will be spotted then.

If the flight plan turns out to be unsafe, the navigator returns to her room to do the planning again and come up with an improved flight plan. This will be followed by another check with the directors, just as often as it takes to make the flight plan safe. Then both the navigator and the director will sign the flight plan, after it has been put on the official form by a secretary specially trained to do so.

Since the fuel has to be paid for by the company itself, a courier then has to take the flight plan to one of the company's logistics people (in another building two miles from where the navigator has her room). This person has to sign the flight plan to approve the use of fuel. Of course, he can refuse to sign. In that case, the refusal will be made clear to the navigator and a letter will be sent to the customer. In this letter, the company will send its excuses and explain why no acceptable flight plan could be produced. Of course, "CRASH" hopes to be of better service in the future.

However, if the person in logistics approves, a courier takes the flight plan back. Then the captain of the aircraft has to sign it. This is because she will be responsible for the aircraft every second of the flight. Again,
the flight plan can be refused, with the same consequences as before. If the flight plan is accepted (by signature), the flight plan will be stored in the computer by one of the directors.

After a successful delivery (despite the company's name, most deliveries are!), the customer will also be sent a letter, accompanied by a bill. However, sometimes a crash does occur. Then an apologizing letter is sent to the customer. All letters to customers are composed and sent by a secretary.

Once a flight plan has been "released" for signing by logistics and the plane's captain, the navigator is available for planning another flight.

About the organization: most navigators are captains as well. Therefore all captains and navigators are united in the AIR division. (They say that AIR stands for "Aces with Incredible Reputations"; being humble is not their strength). Extra captains hired from KLM (Kaptains Looking for Money, an agency that "has" freelance pilots/captains) are also part of AIR, albeit temporarily. Ground support by the loadmasters, directors and meteorology people is covered by the SUPPORT division: SUPPort Of Reliable Transport. The logistics and secretary departments are part of CRASH, but since they couldn't come up with a good name, they don't have a group of their own. The couriers are hired from an agency close to the company.

(a) Construct a resource classification of CRASH, distinguishing roles and groups, using the techniques of the book.
(b) Construct a process model of the process sketched above. Define roles, and assign triggers and roles to tasks whenever appropriate.
(c) Analyze the process and investigate possible improvements.
4 Analyzing Workflows

4.1 Analysis Techniques

The introduction or modification of a business process can have far-reaching consequences. Because a process definition is the blueprint of such a process, it is vitally important that it contains no serious errors. The process should also be designed in such a way that the completion times of and capacity required for cases are kept as small as possible. For example, if two tasks can be carried out in parallel, it in general is sensible to ensure that the process allows this. After all, by "parallelizing" tasks, completion times usually can be reduced. Because the process definition is so important, it is useful to analyze it thoroughly prior to its enactment. In doing so, we differentiate between the analysis of (1) the qualitative aspects and (2) the quantitative aspects of workflows. The former mainly concern the logical correctness of the defined process, that is, the absence of anomalies such as "deadlocks" (when a case is "blocked" and no longer proceeds through the process) and "livelocks" (when a case becomes "stuck" in a never-ending loop). The quantitative aspects mainly concern the performance of the defined process. An analysis of the quantitative aspects focuses upon establishing the performance indicators, such as average completion time, level of service, and utilization of capacity.

In this chapter, we shall highlight a number of analysis techniques which can be extremely useful in the context of workflow management (see figure 4.1). We first introduce a simple technique designed to illustrate all the states attainable in a case. We then turn our attention to the errors that can be made when drawing up the definition of a process. We will show that, based upon the structure of the underlying Petri net, we
can decide whether a process definition is correct. In the second part of this chapter, we concentrate upon the analysis of quantitative aspects. Using a number of examples, we show how to improve the performance of existing processes. Finally, we study the subject of capacity planning.

4.2 Reachability Analysis

As we learned in chapter 2, we can define a process in terms of a Petri net. Figure 4.2 shows such a network.

A Petri net and its initial state determines which states are reachable and in what order they can be reached. (As we saw in chapter 2, the state of a Petri net corresponds with the distribution of tokens over places.) We therefore use a Petri net to specify the possible behavior of a modeled process. One way to illustrate the behavior is to draw up a so-called reachability graph.

This is a directed graph consisting of nodes and directed arrows. Each node represents a reachable state and each arrow a possible change of state. To illustrate this, we can examine the Petri net shown in figure 4.2. The possible states of this network are indicated using "triplets" \((a, b, c)\) with \(a\) representing the number of tokens in the place \(claim\), \(b\) the number in \(under\ Consideration\), and \(c\) the number in \(ready\). We therefore show
the initial state illustrated as (3,0,0). The reachability graph derived from this initial state is shown in figure 4.3.

Using this graph, we can deduce that there is a total of ten attainable states. Each node represents one of these. But not each reachable state actually has to occur. The state (1,2,0), for example, is reached only if the transition *record* fires for a second time when the state is (2,1,0). The number of arrows leading from a node indicates how many subsequent possible states there are. If there is more than one outgoing arrow, then the next state is not predetermined. We refer to this situation as a non-deterministic choice. If a node has no arrows leading from it, then it corresponds with an end state. This is a state in which no transition is enabled. The reachability graph in figure 4.3 shows that the Petri net beginning with the state (3,0,0) always results in the end state (0,0,3) after six firings.
We are paying considerable attention to the reachability graph because it embodies the behavior of the process being modeled. By drawing up the reachability graph for a number of cases, we can gain an insight into the operation of the Petri net tool. The fact that, given a diagram like figure 4.2 (that is, a Petri net and its initial state), we can compile a reachability graph, shows that Petri nets are an unambiguous and precise means of description. Because the operation of a Petri net is completely formalized, it therefore is also possible for a computer to construct the reachability graph.

As we saw in chapter 2, we can use Petri nets to describe processes with a repetitive nature. We used the network shown in figure 4.4 to model the traffic lights at the junction of two one-way streets. The two sets of lights operate in such a way that one is always at red. When both sets of lights are red, there is a token in the place \( x \). As soon as one of the lights changes to green, the token disappears from \( x \) and the other set of lights is blocked. Only when both sets have returned to red is the other light able to change to green. Using the reachability graph shown in figure 4.5, we can study whether the traffic lights do indeed operate in a safe way.

Each possible state in this case is represented by a septet. The figures show the number of tokens in \( red1, green1, yellow1, red2, green2, yellow2, \) and \( x \), respectively. An inspection of the reachability graph shows that the traffic lights do indeed operate safely: in every possible state at least one of the sets of lights is red. However we can see that it
Figure 4.6
The two traffic lights now change to green alternately. It is also possible that the first set always changes to green, while the second set remains constantly at red. We can avoid this by ensuring that each set of lights changes to green in turn. Figure 4.6 shows how this can be modeled.

It is easy to work out that the reachability graph associated with figure 4.6 has a total of six states. Just as we can verify the correct operation of traffic lights using the reachability graph, we can use it to determine the correctness of a workflow. Before we go further into checking correctness, we shall look at a number of typical errors that can occur when defining a process.

4.3 Structural Analysis

Before the introduction of advanced information systems—such as workflow systems—business processes generally had a simple structure.
This was mainly due to the fact that a paper document was linked with each case and could physically only be in one place at any one time. The document acted as a sort of token which ensured that tasks were carried out sequentially. As a result of the many developments in information technology, however, it is now possible to arrange processes completely differently. By using databases and networks, information can be shared. Because different people can work on the same case at the same time, it is no longer necessary for tasks to be performed sequentially. Thanks to the "parallelization" of the business process, enormous reductions in completion times can be achieved. In the environment in which a workflow system operates, it therefore is often attractive to carry out tasks in parallel, as far as possible. But the use of sequential, parallel, selective, and iterative routing in the same process can make it very difficult to assess the correctness of the defined process. We can illustrate this using the simple example in figure 4.7.

At first sight, this appears to be a sensible process definition, with two checks being carried out in parallel following the acceptance of a claim. Based upon these checks, either a rejection letter is sent or a payment is made. However, owing to an incorrect combination of parallel and selective routing, errors have crept into this process definition. If check_policy places a token in c5 and check_claim a token in c6, pay will fire. This is the only scenario in which the case is completed correctly. If check_policy places a token in c3 and check_claim a token in c4, then send_letter will fire twice. The consequence is that two tokens appear in end. If check_policy places a token in c3 and check_claim a token in
Figure 4.8 Four flawed situations
then send_letter will only fire once, but one token will remain in c6. The same happens if check_policy places a token in c5 and check_claim a token in c4.

Figure 4.8 illustrates four situations that, as in the previous example, can result in incorrect processes. Using this figure, we can highlight a number of common errors that occur during the definition of a process:

1. **Tasks without input and/or output conditions.** When a task has no input conditions, it is unclear when it may be performed. When a task has no output conditions, it does not contribute to the successful completion of a case and so it can be dropped. Situation A in figure 4.8 contains one task without input conditions (task4) and one without output conditions (task5).

2. **Dead tasks: tasks that can never be carried out.** It is obvious that a process definition containing "dead" tasks is undesirable. In situation B, task2 can never be performed; the same applies to task3 in situation D.

3. **Deadlock: jamming a case before the condition "end" is reached.** If task1 in situation B places a token in one of the two uppermost places, then the case will wait "ad infinitum" for task1. Only if task1 delivers a token directly to the place end will this deadlock be avoided. In situation D a token can be "jammed" waiting for task5.

4. **Livelock: trapping a case in an endless cycle.** In situation C, every case will remain "ad infinitum" in the cycle consisting of task2 and task3. There thus exists iterative routing without an opportunity to escape.

5. **Activities still take place after the condition "end" is reached.** A good process definition has a clear beginning (the condition start) and end (the condition end). Once the condition end is reached, no more tasks should be carried out. In situation C, task2 and task3 will be fired after the condition end is reached. In this way, an infinite number of tokens will reach the place end. This is clearly an undesirable situation.

6. **Tokens remain in the process definition after the case has been completed.** Once a token appears in the place end, all other references to the case must have disappeared. In situation D, if the case is completed as a result of the firing of task1, there will remain a token in one of the places before task3.

The above shows that, without any knowledge of the actual content of the process being defined, we can identify a number of typical errors in a given process definition. These are connected with the routing of cases. In order to computerize the check for these errors, we need a precise notion of correctness.
4.3.1 Soundness

In the remainder of this book, we use the following minimum requirement that every process must meet:

A process contains no unnecessary tasks and every case submitted to the process must be completed in full and with no references to it (that is, case tokens) remaining in the process.

We call a process that fulfills this minimum requirement sound. We shall formulate the soundness property of a process precisely using figure 4.9.

A workflow process defined in terms of a Petri net has a single input place start and a single output place end. Such a Petri net only makes sense if each transition (task) or place (condition) lies on a directed path from start to end. In other words, there should be no "loose" tasks and conditions. Thanks to this requirement, each task (or condition) can be reached from the place start by following a number of arrows, and the place end is always reachable from each task (or condition) by following a number of arrows. A transition that is not on a path from start to end does not contribute to the successful completion of the process or can be activated at any time. In this section, we only consider Petri nets satisfying this requirement. These Petri nets are called workflow nets (WF-nets).

A workflow net satisfies some syntactical requirements. However, it is still possible to have workflow nets that have anomalies such as potential deadlocks and the inability to terminate. Therefore we define a workflow net to be sound if, and only if, it fulfills the following three requirements:

1. For each token put in the place start, one (and only one) token eventually appears in the place end;
2. When the token appears in the place *end*, all the other places are empty; and
3. For each transition (task), it is possible to move from the initial state to a state in which that transition is enabled.

The first requirement means that every case will be completed successfully over a period of time. The second requirement means that once the case is completed, no references to it will remain in the process. If we combine the first two requirements, we come to the conclusion that—based upon the state illustrated in figure 4.9—there exists only one final state: that is, one with precisely one token in the place *end*. The last requirement excludes "dead tasks"; that is, each task can—in principle—be carried out.

The definition of soundness assumes a notion of *fairness*, that is, if a task can potentially be executed, then it is not possible to postpone its execution indefinitely. Consider for example iterative routing. Although, in principle, it is possible to repeat a part of the process infinitely often, we assume that iteration does not necessarily violate the soundness requirement. Similarly, we assume that two tasks cannot "starve" a third task indefinitely. If we would not make this assumption, any process with selective or iterative routing would not be sound.

How can we establish whether a given process corresponds to a sound workflow net? To do this, we must first check whether the Petri net representing the process is a workflow net. This can be checked by examining the structure of the process. Checking whether the process is sound is more involved. We can check the three soundness requirements using a reachability graph starting with the initial state in which there is only one token in the place *start*. To check the last requirement, we examine whether there is for each task a state transition in the reachability graph which corresponds to the firing of that task. The first two requirements are checked by confirming that the reachability graph has only one final state, and that this is one in which there is precisely one token in *end*. The requirements for correctness just formulated therefore can be checked entirely automatically by inspecting the reachability graph.

There are, however, two drawbacks attached to this approach. First, the construction of the reachability graph for large-scale processes can take up a lot of computer time. It therefore is almost impossible to perform this analysis without a computer. Second, the reachability graph provides
little support in repairing a nonsound process definition. Note that the reachability graph is infinite if tokens can accumulate in a place. It is possible to use variants of the reachability graph, such as the so-called coverability graph, which allows for the detection of such unbounded behavior (see appendix). Nevertheless, these "brute force" approaches can be quite time consuming and do not provide good diagnostics.

Fortunately, there are techniques available for Petri nets that do not suffer from these drawbacks. We do not have the space here to discuss these techniques in depth. However, we shall outline two alternative methods of determining whether or not a process is sound. The first method is based on advanced computer support; the second one can be used manually.

4.3.2 Method with computer support
The first method to determine soundness translates the soundness property to two well-known properties which have been investigated for decades. In order to analyze a process defined in terms of a Petri net, we add an additional transition to the network: $t^*$. This has \textit{end} as its input point and \textit{start} as its output point. The net without transition $t^*$ is called the workflow net; the net with this transition is called the short-circuited net. With this addition, the soundness of the workflow net corresponds with two well-known properties: \textit{liveness} and \textit{boundedness} of the short-circuited net. A Petri net is \textit{live} when it is possible to reach—for each transition $t$ and from every state reachable from the initial one—a state in which transition $t$ is enabled. In a live Petri net, therefore, it remains possible to fire every transition an arbitrary number of times. A Petri net is \textit{bounded} when there is an upper limit to the number of tokens in each place. In other words, it is not possible for the number of tokens in a place to rise without limit if the process is started in the initial state. The traffic lights modeled in figures 4.4 and 4.6 are live and bounded.

Liveness and boundedness are properties which have been researched extensively during the past thirty years. As a result, efficient algorithms and tools are available to analyze them. A process is sound if its Petri net, with the additional transition $t^*$, is live and bounded. The correctness of a defined process thus can be verified by using standard tools. For a number of important subcategories—including the so-called free-choice Petri nets—liveness and boundedness of a network can be established in
polynomial time. Thanks to the many results achieved in the field of Petri-net theory, the soundness of a process can hence be determined efficiently. When a process is not sound, diagnostics can be generated that indicate why this is the case.

The above is merely an illustration of the many analysis possibilities offered by the Petri net representation of a given process. For more information, we refer to the appendix of this book and the very extensive literature about Petri nets.

4.3.3 Method without computer support
The translation of soundness to liveness and boundedness allows for the application of efficient analysis techniques. Unfortunately, the translation is not very intuitive and requires computer support to be relevant. Therefore we propose an alternative method which is easy to apply without computer support or deep theoretical knowledge. We add one requirement to "good" workflow nets in addition to soundness: we will require that the workflow nets are also safe, which means that the number of tokens in each place will never be larger than one. (This means that they are bounded by value one.) It is often easy to check if a net is safe by inspection of the net structure. The method is based on an important property that is very easy to understand in an intuitive way:

If we have two sound and safe workflow nets V and W and we have a task t in V which has precisely one input and one output place, then we may replace task t in V by W and then the resulting workflow net is sound and safe again.

In figure 4.10 this replacement is illustrated.

This property is intuitively clear because a sound workflow net behaves like a transition: it consumes one token from its input place and, after a while, it produces one token in its output place. The environment therefore will not discover the replacement of t by W. The safety of the nets is required in order to avoid the situation that in W two or more tokens will be active at the same time, which may violate the soundness of W.

This replacement property is proved in the appendix. Here we focus on the application of this property. The main idea is as follows:

Suppose we have some set of sound and safe workflow nets, called "building blocks," to start with. If it is possible to derive the workflow
If a transition is replaced by a sound workflow net, then the resulting workflow net is also sound (assuming safeness) net under consideration by a sequence of substitutions of nets from this set of building blocks, then we have proved that our net is sound and save as well.

To illustrate this method we start with a small set of nets for which the soundness and safety is obvious. See figure 4.11. The workflow nets correspond to the typical constructs introduced in chapter 2. There are of course other sets of building blocks possible but this set is already quite powerful.

First we show how we can apply the method. Consider the workflow net shown in figure 4.12.

For this net we can find the derivation presented in the subsequent figures. The method starts with the basic building block shown in figure 4.13.

In the first step, the AND construct is applied to put task $b$ in parallel with task $a$. The resulting workflow net is shown in figure 4.14. Note that we simply applied the AND construct shown in figure 4.11 with $x = a$ and $y = b$. 
1. Basic building block

2. Sequence construct

3. Implicit OR-split construct

4. Explicit OR-split construct

5. Explicit OR-join construct

6. Iteration construct

7. AND construct

Figure 4.11
Sound and safe nets
Figure 4.12
A safe and sound process

Figure 4.13
Apply the AND construct to a (Step 1)

Figure 4.14
Apply the explicit OR-split construct to a (Step 2)

Figure 4.15
Apply the sequence construct to a (Step 3)
In the second step, the explicit OR-split construct is applied to \( a \), that is, the explicit OR-split "pattern" shown in figure 4.11 is applied with \( x = a \) and \( y = c \). The resulting workflow net is shown in figure 4.15.

In the third step, we apply the sequence construct: task \( a \) is followed by task \( d \).

Then the sequence construct is applied to \( b \).

In the fifth step an implicit OR-split construct is applied to \( b \) with the addition of task \( f \) as result.

Then the iteration construct is applied to task \( e \). As a result, task \( g \) is added to the workflow net.

Finally the sequence construct is applied to task \( e \). The resulting workflow net shown in figure 4.20 is exactly the process we wanted to construct. Since we just applied the design patterns shown in figure 4.11, this workflow net is guaranteed to be safe and sound.

As we can see there can be more than one derivation for a particular net. In the example we could have interchanged steps 3 and 4. Not all sound and safe nets have a derivation as is shown in the example presented in figure 4.21.

The reason that we cannot find a derivation here is that two paths that originated at one AND-split should come together in the same AND-join.
Figure 4.18
Apply the iteration construct to e (Step 6)

Figure 4.19
Apply the sequence construct to e (Step 7)

Figure 4.20
The complete process
A process that cannot be constructed using the standard constructs shown in figure 4.11. This is not the case in figure 4.21. This example shows that in case we cannot find a derivation for a particular workflow net, it is not allowed to conclude that the net is not sound and safe: the workflow net shown in figure 4.21 is both safe and sound but it is not possible to construct this net using the standard design patterns shown in figure 4.11.

Note that it is always permissible to add a sound and safe net to our collection of building blocks, so also the net shown in figure 4.21. A particular extension of our replacement rules is a rather trivial one: every place (excluding source and sink places) may be replaced by a place and a task for which this place is the input as well as the output place. In figure 4.22 this transformation is represented.

Suppose that we have found a derivation for a net and that we have to modify the net during a design process. If the modifications are only
replacements of tasks by sound and safe building blocks, everything is fine. But suppose that we have to do another modification: is it necessary to find a new derivation from scratch? The answer is no. We may always go back in the derivation and take another sequence of steps from there after which we continue with the rest of the former sequence. To clarify this we note that in each replacement rule treated so far, we replaced one transition by two other ones with exactly one input and one output place (constructs shown in figure 4.11). In each case the number of transitions with one input and one output increased exactly with one. If we identify the replaced transition with one of the new transitions (with one input and one output) then we have to give the other one a new, unique name. So we can characterize each step in a derivation by a triplet: the selected task, the used building block, and the name of the new task. In the derivation shown in figures 4.13 through 4.20, all tasks have a name. In the table 4.1 we represent this derivation in tabular form.

It is easy to verify that the result of this derivation is the net with tasks \(\{a, b, c, d, e, f, g, h\}\) shown in figure 4.20. Note that we do not mention tasks just added for routing purposes, that is, \textit{AND-split}, \textit{AND-join}, and \textit{OR-split} are omitted.

Suppose that we want to extend the workflow nets shown in figure 4.20 with one additional task \(x\) to obtain the workflow net shown in figure 4.23.

Note that task \(x\) is added by introducing an implicit OR-split. As was argued before we can use the former derivation and simply add a

### Table 4.1

<table>
<thead>
<tr>
<th>step</th>
<th>set of tasks</th>
<th>selected task</th>
<th>used block</th>
<th>new task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>AND</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a,b</td>
<td>a</td>
<td>explicit</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>a,b,c</td>
<td>a</td>
<td>sequence</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>a,b,c,d</td>
<td>b</td>
<td>sequence</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>a,b,c,d,e</td>
<td>b</td>
<td>implicit</td>
<td>f</td>
</tr>
<tr>
<td>6</td>
<td>a,b,c,d,e,f</td>
<td>e</td>
<td>iteration</td>
<td>g</td>
</tr>
<tr>
<td>7</td>
<td>a,b,c,d,e,f,g</td>
<td>e</td>
<td>sequence</td>
<td>h</td>
</tr>
</tbody>
</table>
step between 2 and 3 (step 2.5). After this modification we can continue the derivation as before which results in the net with tasks \{a, b, c, d, e, f, g, h, x\} shown in figure 4.23. Table 4.2 shows this derivation. Using this simple technique we can construct a large set of sound and safe workflow nets.

### 4.4 Performance Analysis

As well as the correctness of a defined workflow, we are also interested in its performance. By this, we mean such quantitative aspects as completion times of cases, the number of cases which can be processed per time unit, the utilization of staff, and the percentage of cases that can be completed within a preset, standard time. To gain insight into the performance of a defined workflow, various analysis techniques can be used. The three techniques most commonly used in this respect are as follows:

1. **Markovian analysis.** Based upon a given workflow, it is possible to generate a Markov chain automatically. This can be used to analyze particular aspects of a workflow. Such a chain contains the possible states of a case and the probability of transitions between them. In fact, the Markov chain is a reachability graph with the probability of transitions added to it. These probabilities are determined based upon measured or expected properties of a case type. Various properties can be established using a Markov chain, for example, what are the chances of a case taking a particular route through a process. By expanding Markov chains with cost and time aspects, a range of performance indicators can be generated. The disadvantage of this approach is that not every aspect...
Table 4.2
Each Step in a Derivation (with 2.5)

<table>
<thead>
<tr>
<th>step</th>
<th>set of tasks</th>
<th>selected task</th>
<th>used block</th>
<th>new task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>AND</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a,b</td>
<td>a</td>
<td>explicit</td>
<td>c</td>
</tr>
<tr>
<td>2.5</td>
<td>a,b,c</td>
<td>a</td>
<td>implicit</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>a,b,c,x</td>
<td>a</td>
<td>sequence</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>a,b,c,d,x</td>
<td>b</td>
<td>sequence</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>a,b,c,d,e,x</td>
<td>b</td>
<td>implicit</td>
<td>f</td>
</tr>
<tr>
<td>6</td>
<td>a,b,c,d,e,f,x</td>
<td>e</td>
<td>iteration</td>
<td>g</td>
</tr>
<tr>
<td>7</td>
<td>a,b,c,d,e,f,g,x</td>
<td>e</td>
<td>sequence</td>
<td>h</td>
</tr>
</tbody>
</table>

can be incorporated into the analysis. Markov-chain analysis can also be very time-consuming (if not intractable).

2. Queueing theory. Queueing theory is intended for the analysis of systems in which the emphasis is placed upon such performance indicators as waiting times, completion times, and utilization of capacity. It therefore is quite a logical way to analyze workflows. In a workflow, there may be queues of cases waiting for resources that cannot process a particular inflow of cases immediately. If we are interested in the formation of a single queue for a number of resources of equal value, then we can confine ourselves to a system consisting of one queue. There are many results available for the analysis of a single queue, which are in general simply to apply. If we wish to evaluate the entire workflow, then we need to consider a network of queues. For queueing networks, some questions can be answered by mathematical methods. Unfortunately, many of the assumptions used in queueing theory are not valid for workflow processes. For example, in the presence of parallel routing, it is often impossible to apply the results obtained from queueing theory.

3. Simulation. Simulation is a very flexible analysis technique. It almost always is possible to analyze a workflow using it. In fact, simulation boils down to the following of a path in the reachability graph. In doing so, particular choices are made based upon various probability distributions. Because simulation is nothing more than the repeated execution of a process with the aid of a computer, it is a technique that is accessible to people without a mathematical background. Simulation therefore results in a better insight into the operation of the process being modeled. Because most simulation tools offer an animation option, the workflow can
be tracked graphically. Moreover simulation can be used to answer a wide range of questions. It is also easy to extend a simulation model with a new aspect (for example, faults). However, the establishment and analysis of a model for a detailed simulation can be a time-consuming affair. Moreover, the careful processing of simulation results requires thorough statistical knowledge.

In this book, simulation is the main analysis technique. The reason for confining ourselves just to one analysis technique is that simulation usually is the only tool supported by the workflow management system. And when we examine the analysis techniques used in BPR, we again see that simulation usually is the only tool available for carrying out quantitative analyses. To illustrate the use of an analysis technique like simulation, we use the process definition shown in figure 4.24.

As figure 4.24 shows, the process consists of two tasks to be performed sequentially. The average number of new cases that arrive at the process per hour is 24. The average time between two successive arrivals therefore is 2.5 minutes. The average time required to carry out both task1 and task2 is 4 minutes each. For each task, 2 resources are devoted exclusively to completing the work item associated with it. These therefore are highly inflexible resources, which can work on only one task. Based upon the figures just given, we can calculate that the average level of resource utilization, that is, the number of arrivals per time unit divided by the number that can be served per time unit, is 80 percent: on average, a resource spends 80 percent of its time working on a task for a particular case. The resource is idle for the remaining 20 percent of the time.
We can now ask ourselves what the average completion time for a case is. In order to determine this, we need to know more about the arrival pattern of new cases and the processing time. For the sake of convenience, we shall assume that the interarrival times are distributed in a negative exponential way. On this hypothesis, it can be determined using either simulation or queueing theory that the average completion time is approximately 22.2 minutes. In other words, it takes an average of 22.2 minutes for a case to move from place $c_1$ to place $c_3$. But of these 22.2 minutes, an average of only 8 minutes is spent on actually working on the case. The remaining 14.2 minutes are waiting time. In this case, therefore, the average waiting time is actually longer than the processing time. In fact, this is actually the case in many real-life situations. Consider, for example, the time spent on waiting to see a doctor. In many administrative processes, things can be even worse: actual processing times are only a small fraction of the total completion time.

As indicated in one of the guidelines for developing workflows, it is sensible—where possible—to perform tasks in parallel. Figure 4.25 shows the process that could be used if it were possible to carry out the two tasks for each case simultaneously. In this situation, the average level of resource utilization remains 80 percent—after all, the supply of cases and the average processing time have not changed. However, the average completion time can be significantly reduced in this way. Using simulation, we can show that the average completion time is now approxi-
approximately 15 minutes. By performing tasks in parallel, we can in this instance achieve a considerable reduction in completion time with the same resources.

It can sometimes be useful to combine two tasks into one larger task. Figure 4.26 shows a process in which task1 and task2 have been fused into a single task12. The average processing time for this new task is 7 minutes. We therefore have assumed that it takes 1 minute less to perform the combined task than to carry out the two original tasks. This reduction is explained by the elimination of set-up time. As a result of the shorter processing time, the average level of resource-capacity utilization has fallen to 70 percent. Moreover, the completion time has dropped dramatically, to an average of 9.5 minutes. So for each case there is now an average waiting time of 2.5 minutes. Compared with the original average waiting time of 14.2 minutes, we thus observe a considerable improvement, which is primarily attributable to increased resource flexibility. The new task12 can be performed by each of the 4 resources. In contrast to the previous situation, each of the resources is busy as long as there is a case to be carried out.

To illustrate the positive influence of resource flexibilization, consider the original process shown in figure 4.27. In this process the two tasks again have to be carried out sequentially. However in this case the resources are not linked to a specific task: each can perform both task1 and task2. As a result, the average completion time is only 14.0 minutes. Compared with the original situation, the average waiting time has fallen from 14.2 to 6 minutes.

Thus far we have assumed that the cases are indistinguishable from one another. In other words, we do not know whether the processing of a particular case will take little or much time. Figure 4.28, though, shows...
Figure 4.28
Situation 5

a situation in which we can differentiate between "easy" and "hard" cases. Performing task1 for an easy case takes an average of 2.66 minutes, whereas for a hard case it takes an average of 8 minutes. On average, 25 percent of the cases are classified as hard, 75 percent as easy. In figure 4.28, we have tried to make use of this information. A special resource has been assigned to perform task1 for hard cases. Besides, there is also a special resource to perform task1 for easy cases. The idea is that the total average completion time can be reduced by separating the two flows. This is the principle also known as triage. In this case, however, it has disastrous results: the average completion time rises to no less than 31.1 minutes. So there is considerable worsening of the situation.

There are instances when triage can have a beneficial effect, though. Consider, for example, the "baskets only" checkout in a supermarket. (Triage is a term which existed long before the rise of BPR and WFM. It
is also used to describe the selection and prioritization of war or disaster casualties according to the nature and seriousness of their injuries.) There are two circumstances in which triage can be useful: (1) when the allocation of specialized resources reduces the average processing time, and (2) when small clients no longer have to wait for large ones to be processed, which reduces the overall average waiting time. The reason that triage has a negative effect in figure 4.28 is that the flexibility of the resources is reduced. For example, only one resource can perform task1 for an easy case. This example shows that thorough quantitative analysis is often required to reach a well-considered workflow design.

The introduction of triage in a supermarket (the "baskets only" checkout) usually shortens the overall completion time because those clients with only a few items do not have to wait behind those with a lot of items. In fact, triage operates in this case as a prioritization rule. In general, we find that triage leads to short completion times when easy cases are actually handled earlier than hard ones. If this is not the case, longer completion times will result. However, we can also apply a prioritization rule without using triage (in other words, without introducing a special queue). Figure 4.29 shows a situation in which for each task the easy cases (those with an average processing time of 2.66 minutes) are given priority over the hard ones (those with an average processing time of 8 minutes). With the aid of simulation, we can show that this results in an average completion time of approximately 14 minutes. So prioritization rules can also deliver considerable savings in completion time. Figure 4.30 lists all the situations again in summary.

![Figure 4.29](image)

Situation 6
The above shows that we can use an analysis technique like simulation to support the design of a workflow. Depending upon the workflow's design, we have seen the average waiting time for a case vary from 2.5 minutes (situation 3) to more than 23 minutes (situation 5). Which design is preferable depends upon the circumstances. There are, however, three guidelines that apply in most situations.

1. *When possible, perform tasks in parallel.* The implementation of parallel processing generally results in short completion times.

2. *Strive for high resource flexibility.* Ensure that resources can perform as many tasks as possible. The use of flexible resources results in higher levels of resource utilization and shorter completion times.

3. *When possible, handle cases in order of processing time.* In general, it is sensible to give cases that have a short processing time priority over those with a longer one. This can be done using triage or prioritization rules.

These guidelines illustrate the fact that there are considerable similarities between the structure and management of logistical and production systems. In fact, a workflow system is a logistical management system. It therefore is important that, when designing workflows, one bears in mind the principles, methods, and techniques which have been developed for structuring and managing logistical and production systems.

### 4.5 Capacity Planning

Thus far we always have assumed that the number of resources in each resource class is fixed. In practice, of course, this is not the case. Employees may fall ill, go on vacation, or leave the company. The

<table>
<thead>
<tr>
<th>Situation</th>
<th>Description</th>
<th>Average completion time</th>
<th>Average processing time</th>
<th>Average waiting time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Situation 1</td>
<td>Sequential</td>
<td>22.2</td>
<td>8.0</td>
<td>14.2</td>
</tr>
<tr>
<td>Situation 2</td>
<td>Parallel</td>
<td>15</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Situation 3</td>
<td>Composition</td>
<td>9.5</td>
<td>7.0</td>
<td>2.5</td>
</tr>
<tr>
<td>Situation 4</td>
<td>Flexibilization</td>
<td>14.0</td>
<td>8.0</td>
<td>6.0</td>
</tr>
<tr>
<td>Situation 5</td>
<td>Triage</td>
<td>31.1</td>
<td>8.0</td>
<td>23.1</td>
</tr>
<tr>
<td>Situation 6</td>
<td>Prioritization</td>
<td>14.0</td>
<td>8.0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Figure 4.30
A summary of the performances in the six situations described
number of staff may also vary according to seasonal factors. Consider, for example, travel insurance sales, which are clearly subject to seasonal influences. This needs to be taken into account when establishing staff allocation. In certain industries we also observe that the supply of new cases follows a clear pattern each week. So the capacity plan is always based upon a particular capacity requirement. The plan shows what resources, and of which type, are needed for each period. Capacity planning may be both short term and long term. In the short term, such factors as sick leave, small fluctuations in the supply of work, days off, overtime, and the hiring of temporary staff play an important role. In the longer term, demand forecasts, seasonal influence, machinery purchases, and staff recruitment policy enter the picture.

If we have a forecast of the supply of new cases, it is easy to estimate the capacity requirement. To illustrate this, we shall use a variant on the process handle complaint introduced in the previous chapter. Figure 4.31 shows the average processing time for each task.

It is assumed that the time taken to perform those tasks that require no resources is negligible. For the others, the average processing time in minutes is shown. For example, the task assess takes an average of 20 minutes. In general, 63% of the cases have been assessed positively at the end of this task, and 27% negatively. In the remaining 10% of cases a further assessment is required. Note that task assess may be executed an arbitrary number of times. The average number of times that assess is executed per complaint is 1.111 (see section 4.5.1). Eventually 70% are assessed positively, and 30% negatively. If we assume that 50 new cases...
<table>
<thead>
<tr>
<th>Task</th>
<th>Average number per day</th>
<th>Average processing time</th>
<th>Average number of minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>contact_client</td>
<td>50</td>
<td>10</td>
<td>500</td>
</tr>
<tr>
<td>contact_dept.</td>
<td>50</td>
<td>15</td>
<td>750</td>
</tr>
<tr>
<td>collect</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>assess</td>
<td>56</td>
<td>20</td>
<td>1111</td>
</tr>
<tr>
<td>pay</td>
<td>35</td>
<td>10</td>
<td>350</td>
</tr>
<tr>
<td>send_letter</td>
<td>15</td>
<td>25</td>
<td>375</td>
</tr>
<tr>
<td>file</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 4.32**
The capacity required per task

Arrive each day, then we can calculate the capacity requirement for each task. Figure 4.32 shows that *assess* requires the most capacity.

A case is assessed an average of 1.111 times, because 10% of them require a second assessment. From an input of 50 cases, therefore, an average of approximately 56 assessments is required. The capacity requirement per task is easy to calculate in this case. In more extensive processes with a large number of iterations, this can be rather more complicated. Fortunately, based upon the process definition it is possible to automatically generate a Markov chain to calculate the capacity requirement for each task.

Based upon the capacity requirement per task, we can calculate the capacity requirement of each resource class. After all, we know from which resource class a required resource will come. As mentioned in the previous chapter, there are four resource classes in this case: Employee, Assessor, Complaints, and Finances. A resource belongs either to Complaints or to Finances, but not to both. Each resource that belongs to the resource class Assessor is automatically a member of the resource class Employee. The task *pay* is the only one requiring a resource from the resource class Finances. The other tasks always require a resource from the resource class Complaints. Moreover, the task *assess* is the only one that requires a resource from the resource class Assessor. Based upon this information, figure 4.33 shows the capacity requirement per resource class.

Figure 4.33 also shows the number of resources required at two particular levels of capacity utilization. When this is 80%, the complaints
Figure 4.33
The capacity requirement per resource class

department requires 8 people. Of these, at least 3 must be assessors. Because resource classes overlap, we must interpret the figures in figure 4.33 carefully. For example, every resource in the resource class Assessor also belongs to the resource class Employee. However, the figures in the row for the category Employee only refer to those employees who do not work as assessors. If we compare the numbers in figure 4.33 with the resources specified in the previous chapter, we see that the complaints department is understaffed for an inflow of 50 cases per day. On the other hand, the finance department has excess capacity.

4.5.1 Method to calculate capacity requirement
For figure 4.31 it is straightforward to calculate the capacity requirements listed in figures 4.32 and 4.33. For complex workflow processes this may be more involved. Therefore we provide more concrete guidelines. To determine the capacity required it is important to know the average number of times each task is executed. In figure 4.31 the tasks record, contact_client, contact_department, collect, and file are executed precisely one time. Task pay is executed 0.7 times, task send_letter is executed 0.3 times, and task assess is executed 1.111 times on average. How to calculate the average number of times each task is executed? One way is to construct a Markov chain that is isomorphic with the reachability graph and add the appropriate cost functions. The drawback of this approach is that the construction of such a Markov chain requires computer support and may be time-consuming. There is also a more pragmatic approach based on the design patterns described in figure 4.11. These patterns can be used to construct safe and sound workflow nets. However, as figure 4.34 shows, the patterns can also be used to determine the average number of times each task is executed.

<table>
<thead>
<tr>
<th>Resource class</th>
<th>Average number of minutes</th>
<th>Number of resources at 80% of capacity</th>
<th>Number of resources at 60% of capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>1975</td>
<td>5.14</td>
<td>6.86</td>
</tr>
<tr>
<td>Assessor</td>
<td>1111</td>
<td>2.90</td>
<td>3.86</td>
</tr>
<tr>
<td>Complaints</td>
<td>2736</td>
<td>7.13</td>
<td>9.50</td>
</tr>
<tr>
<td>Finances</td>
<td>350</td>
<td>0.91</td>
<td>1.22</td>
</tr>
</tbody>
</table>
Analyzing Workflows

Figure 4.34
The number of times each task executed relative to the number of times task \( x \) is executed in the original situation
Compared to figure 4.11, the design patterns in figure 4.34 have been extended with numbers. Assume that task $x$ is executed $N$ times in the original situation, that is, before applying the pattern. If the sequence construct is used, then both $x$ and $y$ are executed $N$ times in the new situation. If one of the three OR constructs is applied, then $x$ is executed $aN$ times and $y$ is executed $(1 - a)N$ times (on average). Note that $a$ is the probability that $x$ is executed in the new situation. If the AND construct is used, then both $x$ and $y$ are executed $N$ times in the new situation.

The iteration construct is a bit more involved. Let $a$ be the probability that after processing $x$ a new iteration is needed. Using calculus one can calculate that in the new situation $x$ is executed $N/(1 - a)$ times and $y$ is executed $aN(1 - a)$ times. To understand these figures consider the iteration construct in figure 4.34. Let $v$ be the expected number of times $x$ is executed for one case starting in place $p$. Then the following equation should hold: $v = 1 + av$, since it happens once and with probability $a$ we return to place $p$. Solving this equation gives $v = 1/(1 - a)$. Task $y$ is executed $v - 1 = a(1 - a)$ times. Therefore, if place $p$ is marked $N$ times, $x$ is executed $N/(1 - a)$ times and $y$ is executed $aN(1 - a)$ times.

Note that the workflow net shown in figure 4.31 cannot be constructed using the design patterns shown in figure 4.34. The standard iteration construct cannot be used to make the loop involving $c5$ and $assess$. However a similar iteration construct can be added to the list of constructs shown in figure 4.34. If $a$ is the probability that $assess$ is executed again, then the total number of times $assess$ is executed equals $aN(1 - a)$ times.

If the average number of new cases per time unit and the average number each task is executed are known, then the average number of times a given task is executed can be calculated by taking the product of these two figures. If the average processing time and corresponding resource class of each task are known, it is straightforward to derive the total number of capacity per time unit per role (assuming a utilization of 100%).

4.5.2 Some basic queueing theory to take variability into account

Because there are always fluctuations in the supply of cases and the processing times, it is not always possible to make full use of the capacity
A n a l y z i n g  W o r k f l o w s         1 3 1

available. It therefore is not sensible to assume that the resources will be utilized to their full capacity. To illustrate this, let us examine a process consisting of one task. During each time unit, $\lambda$ new cases arrive that need to be processed by one resource. This resource is able to complete $\mu$ cases per time unit. The utilized capacity, $\rho$, of this resource is therefore:

$$p = \frac{\lambda}{\mu}$$

If we assume that processing times and case interarrival times are distributed in a negative exponential way, the average number of cases in progress is $L$, where:

$$L = \frac{\rho}{1 - \rho}$$

The average waiting time, $W$—that is, the completion time minus the processing time—is:

$$W = \frac{L}{\mu} = \frac{\rho}{(\mu - \lambda)}$$

The average system time, $S$—that is, the total completion time (waiting time and processing time)—is:

$$S = W + \frac{1}{\mu} = \frac{1}{(\mu - \lambda)}$$

Say an average of 8 new cases arrive per hour, and that an average of 10 cases can be processed per hour. The capacity utilization is therefore 80% ($\rho = 8/10 = 0.8$). On average, there are 4 cases in progress ($L = 4$) and the average waiting time is 24 minutes ($W = 0.4$ hours). With a capacity utilization of 80 percent, the average completion time is thus 30 (24 + 6) minutes. At a capacity utilization of 95 percent and an average processing time of 6 minutes, the average completion time would rise to no less than 2 hours. This small example shows that when the arrival process is irregular, it is not at all sensible to seek a capacity utilization of more than 80 percent.

Figure 4.35 shows the impact of utilization on the average number of cases in progress. The impact resulting from the duplication of utilization from 0.25 to 0.50 (+0.66 cases) is much smaller than the impact from the small increase from 0.98 to 0.99 (+50 cases).

The situation just described corresponds with the $M/M/1$ queue. The first M shows that the interarrival times are distributed in a negative exponential way. The second M shows that the processing times are also distributed in this way. The number 1 indicates that there is only one
Chapter 4

Table 4.35
The average number of cases in progress given a utilization ratio

<table>
<thead>
<tr>
<th>Utilization (p)</th>
<th>Average number in progress (L)</th>
<th>Utilization (p)</th>
<th>Average number in progress (L)</th>
<th>Utilization (p)</th>
<th>Average number in progress (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>0.11</td>
<td>0.25</td>
<td>0.33</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>0.80</td>
<td>4.00</td>
<td>0.85</td>
<td>5.66</td>
<td>0.90</td>
<td>9.00</td>
</tr>
<tr>
<td>0.95</td>
<td>19.00</td>
<td>0.98</td>
<td>49</td>
<td>0.99</td>
<td>99</td>
</tr>
<tr>
<td>0.9999</td>
<td>999</td>
<td></td>
<td></td>
<td>0.9999</td>
<td>9999</td>
</tr>
</tbody>
</table>

Figure 4.35
The average number of cases in progress given a utilization ratio

resource. To show just how sensitive the waiting times are to the variability of the processing times, we can consider the M/G/1 queue. In this the processing times are distributed randomly (G = general). The only things we know are that the average processing time is $1/\mu$ and that the standard deviation is $a$. Based upon these two parameters, we can define the coefficient of variation, $C$:

$$C = \frac{\mu \lambda}{\rho}$$

The coefficient of variation is a measure of relative deviation from the average. The higher $C$ is, the wider the spread of processing times will be. In the M/G/1 queue, capacity utilization is also equal to $\rho = \lambda/\mu$. However, the average number of cases in progress ($L$) now depends upon the coefficient of variation:

$$L = \rho + \left(\frac{\rho^2}{2(1-\rho)}\right)(1 + C^2)$$

(This is known as the Pollaczek-Khinchin formula.) The average waiting time, $W$, also strongly depends upon the value of $C$:

$$W = \frac{\rho}{(2\mu(1-\rho))}(1 + C^2)$$

These formulae show that large variations in processing times can result in long completion times. Conversely, regular processing times will deliver shorter completion times. To illustrate this, let us assume a situation in which an average of 8 new cases arrive per hour, and the processing time for each is precisely 6 minutes. In this case, the coefficient of variation $C$ is 0. By applying the formulae, we discover that the average waiting time is only 12 minutes. The completion times therefore depend strongly upon the variation in processing times. Note that in case of negative exponentially distributed processing times, $C$ equals 1 and the Pollaczek-Khinchin formula reduces to the formula given earlier.
We have just made use of a number of simple formulae from the queueing theory, part of the discipline of operations research (OR). There are many results from the queueing theory that can be applied directly in the context of workflow management. As well as the $M/M/1$ and $M/G/1$ queues discussed earlier, $M/M/n$ queues (ones containing several identical resources) are also easy to analyze. For $M/G/n$ queues and $G/G/n$ queues, there exist formulae for approximating the average waiting time. One result that is applicable to every queue (regardless of interarrival pattern, distribution of processing times and number of resources) is Little's formula:

\[
L = \frac{\lambda}{S}
\]

This establishes a link between the number of cases in progress, $L$, the intensity of the interarrival process, $\lambda$, and the average system time, $S$. If the average completion time for a case is 5 days ($S = 5$), and an average of 25 new cases arrive per day ($\lambda = 25$), then the average number of cases in progress is 125 ($L = 125$).

Given an expected supply of cases and a number of assumptions about their processing, we can use simulation and/or the queueing theory to determine the capacity requirement during a particular period. Based upon these capacity requirements, a capacity plan can be drawn up. When preparing a capacity plan, fluctuations in case supply, temporary loss of resources, and other problems should also be taken into account. The same applies to the desired level of service. To guarantee short completion times, it is sometimes necessary to substantially increase the number of resources.

There is a clear link between capacity planning in a workflow environment and in a production environment. Many concepts used in manufacturing resources planning (MRP-II) systems can be directly transferred into workflow management systems. Rather than the bill of material (BOM), however, it is now the process definition which is the starting point.

EXERCISES

Exercise 4.1    Optimize data usage
Consider the sequential process modeled in terms of a role/route diagram in figure 4.36.
There are nine tasks and the employees are divided into three resource classes (roles): X, Y, and Z. Each task needs to be executed by someone with the appropriate role.

(a) Model the process definition in terms of a Petri net.
(b) Is the role/route diagram appropriate for the specification of workflow processes?

For the execution of the workflow process the following nine data elements are relevant: \(D_1, D_2, \ldots, D_9\). The relationships between data elements and tasks are given in the CRUD matrix shown in figure 4.37.

Assume that only the data elements and their usage are relevant for the ordering of tasks. The sequential process shown in the role/route diagram is far from optimal, that is, task 4 can be executed directly after task 1; there is no need to wait for task 2 and task 3.
(c) Improve the process by making it more parallel.
(d) Is it a good idea to combine tasks? If so, which tasks are proper candidates?

**Exercise 4.2  Invariants**
Consider the Petri nets shown in figure 4.38, figure 4.39, figure 4.40, and figure 4.41.

Answer for each Petri net the following questions (see appendix A):
(a) What are the place invariants (maximum 5)? What do they show?
(b) What are the transition invariants (maximum 5)? What do they show?
Figure 4.39
Network

Figure 4.40
Network
Figure 4.41
Supply chain
(c) Is the net bounded?
(d) Is the net live?
(e) Is the net free-choice?
(f) What are the S-components?

Exercise 4.3  Verification process definition

Consider the process definition shown in figure 4.42.

(a) Check, by constructing the reachability graph, the correctness.
(b) Estimate the number of states when condition c6 is removed.
(c) Prove by place invariants that the two sub-procedures (t2... t6 and t1... t12) are not active at the same time (mutual exclusion).
(d) Prove that there is a linear dependency between start and ready (give conservation laws in terms of place invariants).

Exercise 4.4  Search for errors

Consider the process definitions shown in figures 4.43, 4.44 and 4.45.

Answer for each process definition the following questions:

(a) Is the process definition correct?
(b) If not, show the error (reachability graph and/or place invariants).

Exercise 4.5  Performance analysis I

Consider the process in figure 4.46.

(a) Determine the following performance indicators:
• Occupation rate (utilization) for each resource,
• Average WIP (work in progress),
• Average flow time (throughput time), and
• Average waiting time for each task.

Task 2 is a check task. The management thinks about a selective execution of this task where only 25% of the cases are checked. The average service time of this new task is 6 minutes.

(b) Determine the performance indicators again:
• Occupation rate (utilization) for each resource,
• Average WIP (work in progress),
• Average flow time (throughput time), and
• Average waiting time for each task.
Figure 4.42
Network Analyzing Workflows 139

[Diagram of a network with labeled nodes and connections]
Exercise 4.6  Performance analysis
Consider the process in figure 4.47.

(a) Determine the following performance indicators:
- Occupation rate (utilization) for each resource,
- Average WIP (work in progress),
- Average flow time (throughput time), and
- Average waiting time for each task.

The two resources working on task 1 join forces and work together on both easy and difficult cases. As a result the average time to handle task 1 for one case is two minutes (i.e., a total of 4 minutes of capacity).
(b) Determine the performance indicators again:
- Occupation rate (utilization) for each resource,
- Average WIP (work in progress),
- Average flow time (throughput time), and
- Average waiting time for each task.

Exercise 4.7    Performance analysis III
Consider a process in which \( ct1 \) and \( ct2 \) are checks (see figure 4.48). If they are positive, task \( bt \) (e.g., pay damage) is executed. If one of them is negative, \( bt \) is skipped. The two check tasks are independent of each other.
(a) Determine the following performance indicators:
• Occupation rate (utilization) for each resource,
• Average WIP (work in progress),
• Average flow time (throughput time), and
• Average waiting time for each task.

Give at least two alternatives, that is, improved workflow definitions.

(b) For each alternative answer the following questions:
• Why is it better?
• What is the utilization of resources?
• What is the maximal throughput?
Analyzing Workflows

**Figure 4.49**
Workflows

**Figure 4.50**
Client/server
Exercise 4.8    E-business

In electronic business workflows of different organizations are coupled. One of them plays the role of client and the other of server. These workflows are shown in figure 4.49.

(a) Give derivations for the client and the server.
(b) Use these derivations to obtain the derivation of the coupled workflows. (Herewith we have proven that this coupling is sound and safe)

In figure 4.50 we see again the coupling between two processes: a client process and a server process. During the course of the server process there is some exchange of information between the server and the client: after task $d$ has been done, a message is sent from $t$ to $q$ and later, when task $c$ is done a message is sent from $r$ to $v$.

(c) Is there a derivation with building blocks replacement possible?
(d) Is it a sound and safe workflow? Give arguments.