Chapter 2
Process-Aware Information Systems

Abstract The success of commercial enterprises increasingly depends on their ability to flexibly and quickly react to changes in their environment. Businesses are therefore interested in improving the efficiency and quality of their business processes and in aligning their information systems in a process-centered way. This chapter deals with basic concepts related to business process automation and process-aware information systems (PAISs). Characteristic properties, perspectives, and components of a PAIS are presented based on real-world process scenarios.

2.1 Introduction

As discussed in Chap. 1, businesses are increasingly interested in improving the quality and efficiency of their processes, and in aligning their information systems in a process-centered way; i.e., to offer the right *business functions* at the right *time* to the right *users* along with the *information* and the *application services* (e.g., user forms) for them. In this context, process-aware information systems (PAISs) have emerged to provide a more dynamic and flexible support for business processes [82].

To provide additional value for the business, however, any automation of its processes through PAISs should be preceded by process re-engineering and optimization efforts [130, 273]; i.e., business processes have to be (re-)designed to meet organizational goals in an economic and efficient manner. Frequent goals pursued in this context are shortening process cycle times, reducing process costs, increasing customer satisfaction by utilizing available resources in an efficient and effective manner, and decreasing error rates.

To discuss alternative process designs with stakeholders and to evaluate the designed processes in respect to these goals, process knowledge must be first captured in *business process models*. These models describe business processes at a rather high level of abstraction and serve as a basis for process analysis, simulation, and visualization. More precisely, a business process model should comprise all the activities of a business process and their attributes (e.g., cost and time) as well as the control and data flow between the activities. The activities in a business process, in turn, may be *manual* ones without the potential to be automated and hence lying outside the scope of a PAIS, or *system-supported* ones requiring human or machine resources for their execution [367]. Hence, a business process model does not always correspond to the part of the business process automated in a PAIS.

Accordingly, a distinction has to be made between business process models on one hand, and their executable counterparts (also denoted as *executable process models* or *workflow models*) on the other [240]. The latter can realize the automation of business processes and thus, in whole or part, the implementation of their models. When interpreting an executable process model, documents, data objects, or activities are passed from one process participant to another according to predefined procedural rules [367].

Modern PAISs describe process logic explicitly in terms of such executable process models which provide the schema for process enactment. This book focuses on executable process models and their flexible support through PAISs, whereas issues related to the modeling, analysis, and redesign of business processes are outside the scope of this book. When realizing a PAIS based on executable process models, however, one has to bear in mind that there is a variety of processes showing different characteristics and needs. On one hand, there exist well-structured and highly repetitive processes whose behavior can be fully *prespecified*. On the other hand, many processes are knowledge-intensive and highly dynamic. Typically, the latter cannot be fully prespecified, but require *loosely specified* process models. Using real-world process scenarios, this chapter gives insights into these different process categories. Furthermore, it presents properties, perspectives, and components of PAISs. Thereby, it abstracts from the subtle differences that exist between the different PAIS-enabling technologies and process modeling paradigms, focusing instead on core features.

The chapter is organized as follows: Section 2.2 presents examples of prespecified and repetitive processes, and discusses their basic properties, while Sect. 2.3 introduces examples of knowledge-intensive and dynamic processes to illustrate the broad spectrum of business processes to be supported by PAISs. Following this, Sect. 2.4 deals with the different perspectives on a PAIS, and Sect. 2.5 presents its build- and run-time components. Section 2.6 concludes this chapter with a short summary.

2.2 Prespecified and Repetitive Processes

This section deals with repetitive business processes whose logic is known *prior* to their execution and can therefore be prespecified in process models [178]. After emphasizing the need for the IT support of business processes, we provide two realistic examples of prespecified processes from the health care domain.

2.2.1 Motivation

To understand why IT support for repetitive business processes is needed, we first describe a medical scenario (cf. Example 2.1).

Example 2.1 (Need for Supporting Prespecified Processes). The work of a hospital's medical staff is significantly burdened by organizational tasks [172]. Medical procedures (e.g., lab tests and diagnostic imaging) have to be planned and prepared, appointments with different service providers (e.g., cardiology or radiology) scheduled, lab specimens or the patients themselves transported, visits of physicians from other departments arranged, and medical reports written, transmitted, and evaluated. Thus, cooperation between the medical staffs of different departments is a repetitive, but crucial task. Still, in many hospitals such organizational tasks have to be coordinated manually by staff members requiring, for example, time-consuming phone calls and documentation steps. In practice, this often leads to organizational problems and to high administrative loads for staff members resulting in numerous errors and undesired delays. Patients may have to wait because resources (e.g., physicians, rooms, technical equipment) are not available. Medical procedures may be impossible to perform if information is missing, or preparatory work is omitted, postponed, canceled, or requires latency time. Previously made appointments may then have to be rescheduled, again resulting in phone calls and a loss of time. Moreover, insufficient communication and missing process information are among the major factors contributing to adverse events in a medical environment (e.g., complications due to incorrect treatment). For these reasons hospital stays are often longer than required, and costs or invasive treatments unnecessarily high.

From this scenario, it becomes clear why the demand for effective IT support of repetitive processes for businesses is growing, especially in hospitals. In this context, a PAIS can foster the collaboration and communication among staff members and contribute reducing the number of errors by selectively providing accurate and timely information to process participants. To further illustrate this, Sect. 2.2.2 presents two examples of processes that are relevant in the context of the scenario described in Example 2.1.

2.2.2 Examples of Prespecified Processes

This section introduces two real-world examples of prespecified processes from the health care domain [172, 267]. The first one deals with medical order entries and the respective reports generated by them (cf. Example 2.2). This process is characteristic for many hospitals and enables the coordination of interdepartmental communication between a ward and a radiology department.

Example 2.2 (Medical Order Entry and Result Reporting). Consider Fig. 2.1. It shows the process of handling a medical order between a ward and the radiology department. First, the medical order is placed by the ward's nurse or a physician; this is accomplished by performing consecutively activities *Prepare Order* and *Enter Order*. Following this, the order is sent to the radiology department. When receiving the order, this department checks the medical indication to decide whether or not the order can be accepted (activity *Check Indication*). Depending on the outcome of this activity, either the order is rejected (activity *Reject Order*) or the requested X-ray is scheduled (activity *Schedule X-rays*). In both cases, the ward is informed accordingly. If the order is rejected, the process will be completed. Otherwise, at the scheduled date, the patient is sent to the radiology department (activity *Transfer Patient*), where the X-rays are performed by a radiographer (activity *Perform X-rays*), and the resulting images are diagnosed by the radiologist in the examination room (activity *Diagnose Images*). Afterward the radiology report is created (activity *Create Report*) and its validity is checked (*Check Validity*). If required, iterations for corrections may be performed before a senior radiologist validates the report (*Validate Report*). The final report is then printed (*Print Report*) and signed (*Sign Printed Report*), before it is sent to the ward. When the ward receives the report (*Receive Report*) the process is completed.

The *prespecified process model* from Fig. 2.1 reflects the activities to be performed and the control flow between them. For example, the process model comprises the two activities *Prepare Order* and *Enter Order* as well as the precedence relation between them (represented by the arc linking these two activities). The latter indicates that activity *Prepare Order* has to be completed before activity *Enter Order* may be started. Furthermore, the process model contains alternative execution paths; e.g., after completing activity *Check Indication* either *Reject Order* or *Schedule X-rays* is performed. Finally, the depicted process model also refers to different organizational units, i.e., a *ward* and the *radiology department*.

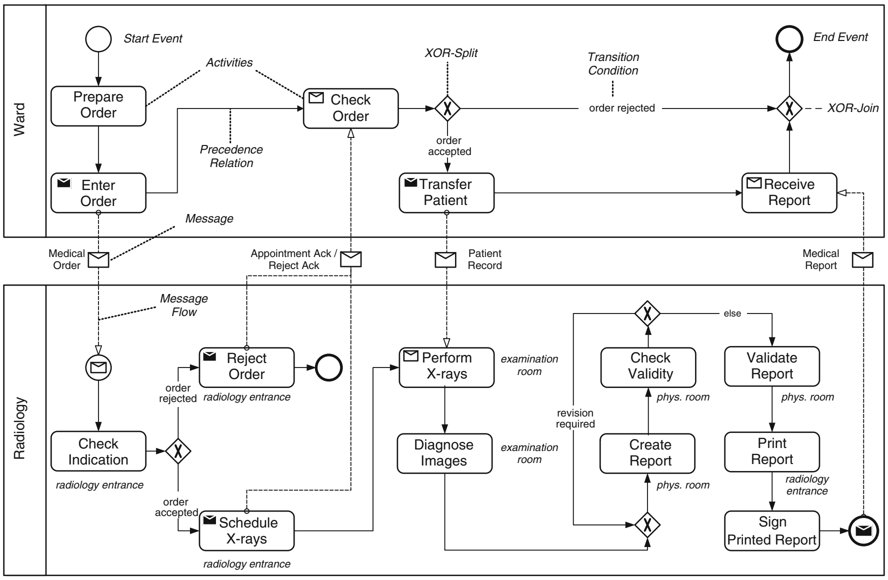

Fig. 2.1 Process for medical order handling and result reporting

Our second process example deals with laparoscopic surgery in a woman's hospital (cf. Example 2.3). This process is highly repetitive and provides routine medical services like cyst removal, dilation, or curettage. Furthermore, it is well structured and involves three organizational units: the outpatient department, the surgical ward, and the surgical suite.

Example 2.3 (Planning and Performing a Laparoscopic Surgery). Consider the prespecified process from Fig. 2.2. It starts with the referral of a patient to the hospital. The patient is first admitted to the outpatient department (1) where a medical checkup is performed (2), a medical examination takes place (3), and then anesthesia and its attending risks are discussed with the patient (4 and 5). Following this, a decision is made (6). If there are contraindications, a discharge letter will be written (7) and the process is completed. Otherwise a date for the surgery is scheduled (8), before the patient may return home. Before the day of the surgery, a physician from the surgical ward to which the patient will be admitted checks the patient's medical record (9). On the scheduled date, the patient is admitted to the surgical ward (10), prepared for the surgery (11), and sent to the surgical suite (12), where the surgery is then performed (13). Afterward, the patient is transferred back to the surgical ward (14), where postsurgical care is provided (15). Usually, the patient is discharged on the same day (18). Meanwhile, the doctor from the surgical

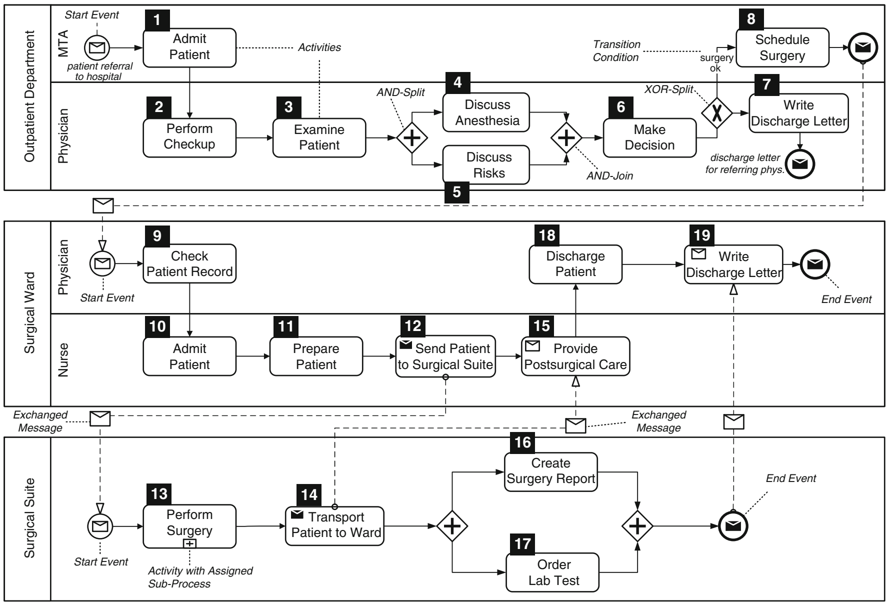

Fig. 2.2 Prespecified process for planning and performing a Laparoscopic surgery

suite orders a lab test (17) and creates a surgery report (16). Based on this, a discharge letter is created (19) and sent to the physician who referred the patient to the hospital.

The prespecified process model from Fig. 2.2 starts with an incoming message on the referral of a patient and ends with an outgoing message referring to the discharge letter. Furthermore, the process model includes activities that may be executed in parallel. For example, this applies to activities *Create Surgery Report (16)* and *Order Lab Test (17)*. Finally, for each human activity, the prespecified process model contains information about the potential actors performing this activity. In our example, this is accomplished by referring to entities from the *organizational model* of the hospital; e.g., activity *Examine Patient (3)* may be performed by any user designated *Physician* who belongs to the *Outpatient Department*.

2.2.3 Discussion

Processes as illustrated in Examples 2.2 and 2.3 can be found in every hospital. Particularly, they are highly repetitive and represent the organizational knowledge necessary to coordinate daily tasks among staff members and across organizational units. Although the described processes are long-running (up to several days, weeks, or months), they are well structured; i.e., process logic is known *prior* to the process execution and thus can be prespecified in *process models* like the ones depicted in Figs. 2.1 and 2.2. Generally, such a *prespecified process model* captures the activities to be executed, their control and data flow, the organizational entities performing the activities, and the data objects and documents accessed by them. Note that at this stage, it is not important to fully understand the notation used in Figs. 2.1 and 2.2 (cf. Chap. 4), but to recognize that there are existing processes whose logic is usually known prior to their execution and thus can be prespecified in the respective process models. Prespecified processes and their support, along the different phases of the process life cycle, will be discussed in *Part II (Flexibility Support for Prespecified Processes)* of the book, i.e., in Chaps. 4–10.

2.3 Knowledge-Intensive Processes

2.3.1 Motivation

In practice, many business processes are *knowledge-intensive* and highly *dynamic*; i.e., the process participants decide on the activities to be executed as well as their order. Typically, respective processes cannot be fully prespecified like the ones described in the previous section. Example 2.4 emphasizes this in a clinical scenario.

Example 2.4 (Complex Patient Treatment Scenario). When being confronted with a complex patient treatment case, physicians have to dynamically decide which examinations or therapies are necessary (e.g., taking costs and invasiveness into account) or even dangerous due to contraindications and treatment-typical problems. Further, many procedures require a preparation; e.g., before a surgery can take place a patient has to undergo preliminary examinations, either of which requires additional preparation or excludes other interventions. While some of them are known in advance, others have to be scheduled dynamically. Generally, decisions about the next process steps have to be made during treatment by interpreting patient-specific data according to medical knowledge and by considering the current state of the patient. It is generally agreed that such knowledge-intensive processes cannot be fully automated [172]; i.e., physicians should not blindly obey any arbitrary step-by-step treatment plan (i.e., prespecified process model), but need to provide the best possible treatment for their patients after taking into account the given situation.

Supporting such complex scenarios requires loosely specified processes (e.g., medical guidelines providing assistance in the context of a particular medical problem) as the one described in Example 2.5. Furthermore, as will be shown in Example 2.6, in many scenarios it is even not possible to straightjacket a process into a set of activities, but to explicitly consider the role of business data as a driver for flexible process coordination.

2.3.2 Examples of Knowledge-Intensive Processes

We introduce two examples of knowledge-intensive processes. While the first one can be directly related to the scenario from Example 2.4 and corresponds to a loosely specified process, the second example emphasizes the role of data as a driver for process execution.

2.3.2.1 Loosely Specified Processes

Example 2.5 deals with a simplified medical guideline we adopted from [19]. It describes a constraint-based process of treating a patient admitted to the emergency room of a hospital suspected of having a fracture (cf. Fig. 2.3).

Example 2.5 (Fracture Treatment Process). Consider Fig. 2.3. Before any treatment may be chosen, activity *Examine Patient* has to be performed by a physician (constraint *init*). If required, additional medical diagnosis is done by executing activity *Perform X-rays*. Depending on the presence and type of fracture, four different treatments exist: *Prescribe Sling*, *Prescribe Fixation*, *Perform Surgery*, and *Apply Cast*. Except for *Apply Cast* and *Prescribe Fixation*, which are mutually exclusive (constraint *not co-existent*), the treatments can be applied in any combination and each patient receives at least one of them (*1-of-4 constraint*). Activity *Perform X-rays* is not required if the specialist diagnoses the absence of a fracture when performing activity *Examine Patient*. If activity *Perform X-rays* is omitted, only the treatment *Prescribe Sling* may be applied. All other treatments require *Perform X-rays* as preceding activity in order to rule out the presence of a fracture, or to decide how to treat it (constraint *precedence*). Simple fractures can be treated just by performing activity *Apply Cast*. For unstable fractures, in turn, activity *Prescribe Fixation* may be preferred over activity *Apply Cast*. When performing activity *Perform Surgery*, the physician is further advised to (optionally) execute activity *Prescribe Rehabilitation* afterward

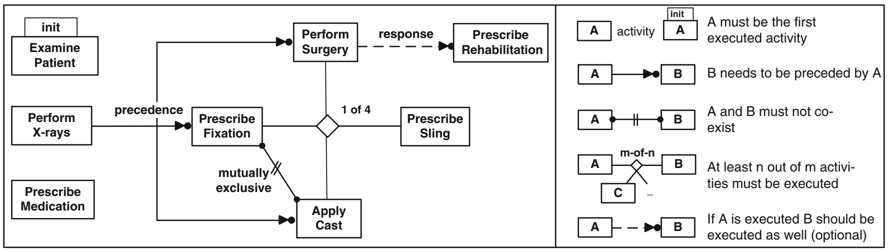

Fig. 2.3 Example of a constraint-based process (adopted from [19])

(optional constraint *response*). Moreover, the physician may execute activity *Prescribe Medication* (e.g., pain killers or anticoagulants) at any stage of the treatment. Note that activities *Examine Patient* and *Perform X-rays* may be also performed during treatment.

Figure 2.3 depicts the *loosely specified process model* corresponding to Example 2.5 when using a *constraint-based process modeling* approach. The boxes represent *activities* and the relations between them are different kinds of *constraints* for executing these activities. The depicted model contains *mandatory constraints* (solid lines) as well as one *optional constraint* (dashed line). As opposed to fully prespecified process models (cf. Figs. 2.1 and 2.2) that describe *how* things have to be done, constraint-based process models (cf. Fig. 2.3) only focus on the logic that governs the interplay of actions in the process by describing the *activities* that can be performed and those *constraints* prohibiting undesired behavior.

Note that in more complex cases, the physician in charge may have to choose from dozens or even hundreds of activities. While some of them may be executed any number of times and at any point in time during the treatment process, for others a number of constraints have to be obeyed; e.g., certain activities may have to be preceded or succeeded by other activities or may even exclude certain activities. Moreover, depending on the particular patient and his medical problems, certain activities might be contraindicated and should therefore not be chosen. The challenge is to provide PAIS support for such knowledge-intensive processes and to seamlessly integrate the described constraints within the physician's work practice. Generally, the structure of knowledge-intensive processes strongly depends on user decisions made during process execution; i.e., it *dynamically* evolves.

Loosely specified processes will be discussed in *Part III (Flexibility Support for Loosely Specified Processes)* of the book, i.e., in Chaps. 11–12.

2.3.2.2 Data-Driven Processes

Both prespecified and loosely specified processes are *activity-centric*; i.e., they are based on a set of activities that may be performed during process execution. In practice, however, many knowledge-intensive and dynamic processes exist that cannot be straightjacketed into a set of activities [24, 159]. Characteristic of these processes is the role of business data acting as a driver for process execution and coordination. Capturing the logic of respective processes in activity-centric process models like the ones depicted in Figs. 2.1–2.3, therefore, would lead to a "contradiction between the way processes can be modeled and preferred work practice" [315].

In the area of human resource management, for instance, recruitment constitutes an example of a knowledge-intensive, data-driven process [156, 160]. It starts with the human resource department receiving recruitment requisitions from a department of the organization, followed by steps like preparing the job description, identifying the employees to be involved (e.g., advertising and interviewing activities), finding the best candidates among the applicants, arranging and conducting the interviews with the selected candidates, and making the final decision. Example 2.6, which we adopted from [160], describes such a recruitment process.

Example 2.6 (Recruitment Process). Applicants may apply for *job vacancies* via an Internet form. The overall process goal is to decide which applicant shall get the job. Before an applicant can send her *application* to the respective company, specific information (e.g., name, e-mail address, birthday, residence) has to be provided. Once the *application* has been submitted, the responsible officer in the human resource department is notified. Since many applicants may apply for a *vacancy*, usually, a number of various personnel officers might be handling the *applications*. If an *application* is ineligible, the applicant is immediately rejected. Otherwise, personnel officers may request internal *reviews* for each applicant. Depending on the respective divisions of the company involved, the concrete number of *reviews* may vary from *application* to *application*.

Corresponding *review* forms have to be filled in by employees from the various company divisions before a stated deadline. Employees may either decline or accept performing the requested *review*. In the former case, they must provide a reason; otherwise, they propose how to proceed; i.e., they indicate whether the applicant shall be invited for an *interview* or be rejected. In the former case, an additional appraisal is needed. After the employee has filled in the *review* form, she submits it to the personnel officer. In the meanwhile, additional *applications* may have arrived; i.e., different *reviews* may be requested or submitted at different points in time. In this context, the personnel officer may flag already evaluated *reviews*. The processing

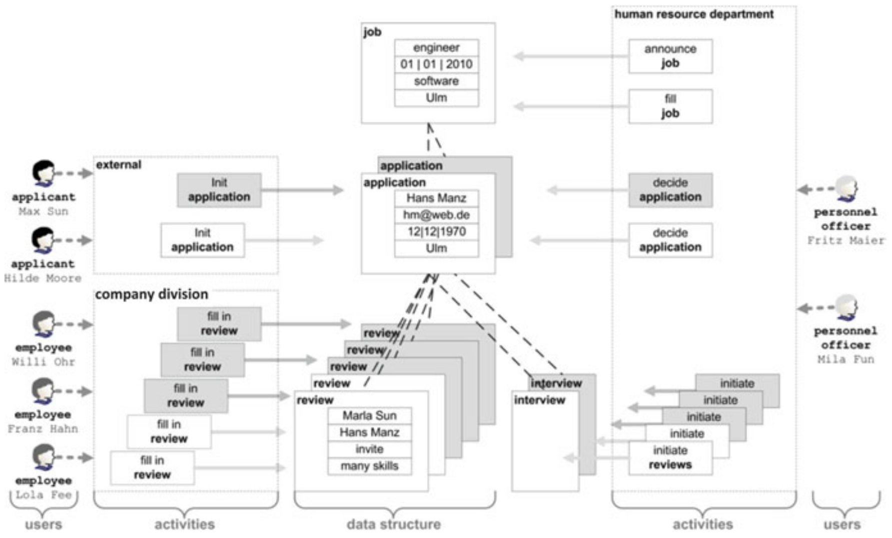

Fig. 2.4 Example of a data-driven process (adopted from [160])

of the *application* proceeds while corresponding *reviews* are created; e.g., the personnel officer may check the CV and study the cover letter of the *application*. Based on the incoming *reviews* he makes a decision about the *application* or initiates further steps (e.g., *interviews* or additional *reviews*). Finally, he does not have to wait for the arrival of all *reviews*; e.g., if a particular employee suggests hiring the applicant. An illustration of this example is provided by Fig. 2.4.

Example 2.6 describes a scenario in which *business data* acts as a driver for process execution; i.e., *objects* (e.g., applications and reviews), *object attributes*, and *object relations* play a fundamental role for process execution. Therefore, a tight synchronization between the object and process state is required; i.e., it is no longer sufficient to only model processes in terms of black-box activities. Instead, their execution is related to *objects* and *object states* [158]. Unlike activity-centric approaches, enabling a particular process step does not directly depend on the completion of preceding process steps, but rather the changes of certain object attribute values.

Data-driven processes will be discussed in *Part IV (User- and Data-Driven Processes)* of the book, i.e., in Chaps. 13–14.

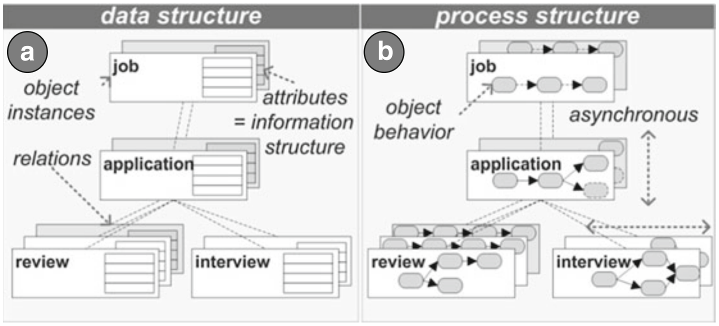

Fig. 2.5 Relation between data and process modeling (adopted from [160])

2.3.3 Discussion

Typically, knowledge-intensive processes are very dynamic. In particular, the concrete activities to be performed, as well as their exact course, may depend on decisions made by process participants during process execution; i.e., knowledge-intensive processes cannot be completely prespecified. Instead, more maneuvering room for process participants and looseness of process execution are required. In certain cases this can be achieved by using constraint-based process models focusing on *what* shall be done by describing the activities that may be performed as well as related constraints prohibiting undesired process behavior.

As further shown, there are scenarios in which an activity-centric approach is not suitable at all, but a tighter integration of process and data is required. In accordance to data models comprising object types and object relations (cf. Fig. 2.5a), the modeling and execution of processes have to be based on two levels of granularity: *object behavior* and *object interactions* (cf. Fig. 2.5b).

The scenarios presented in Examples 2.2–2.6 indicate that a variety of business processes exist for which PAIS support is needed. Effective IT support for this wide spectrum of business processes necessitates different approaches considering the specific properties of prespecified, loosely specified, and data-driven processes. Later chapters of this book will introduce dedicated approaches supporting processes of the different categories in a flexible manner. For the remainder of this chapter, however, we abstract from the subtle differences existing between the these process support paradigms and focus on basic commonalities and notions of PAISs.

2.4 Perspectives on a PAIS

Generally, different perspectives on a PAIS can be taken (see [138] for a corresponding framework). For example, relevant perspectives include function, behavior, information, organization, operation, and time. As illustrated by Fig. 2.6, operational

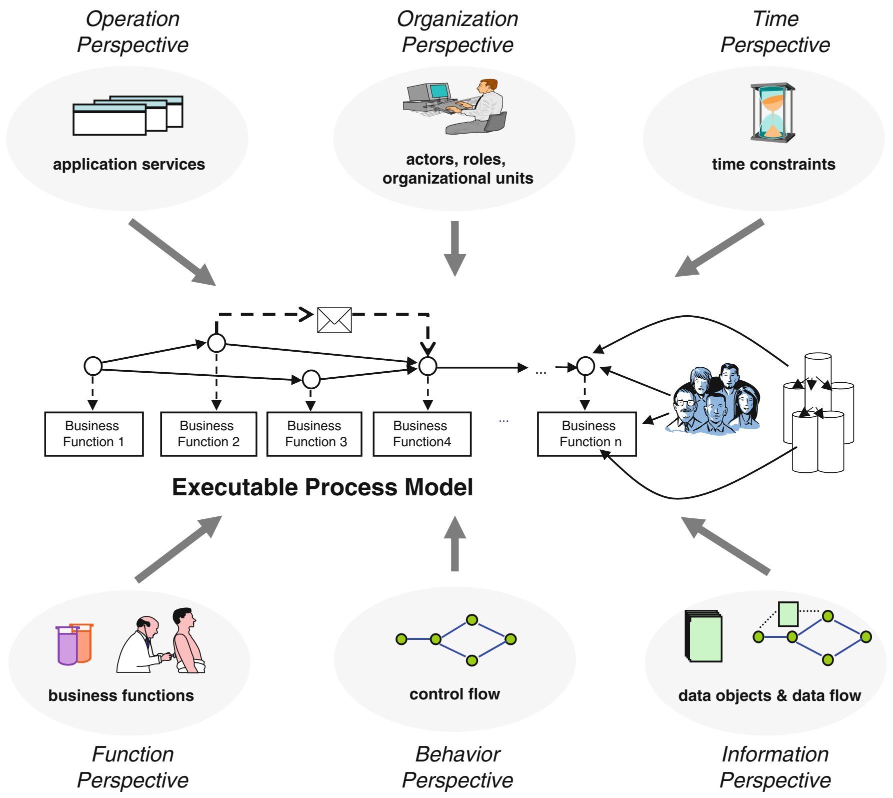

Fig. 2.6 Perspectives in a PAIS

support of these perspectives and their integration with executable process models are needed in order to offer the right business functions at the right time and to the right users along with the information and the application services needed. In the following, the different perspectives are presented in detail.

2.4.1 Function Perspective

The *function perspective* covers the functional building blocks from which activity-centric process models can be composed; i.e., *atomic activities* representing elementary business functions as well as *complex activities* representing subprocess models. To be more precise, an atomic activity constitutes the smallest unit of work, i.e., a description of a business function that forms one logical step within an executable process model. Usually, atomic activities require human or machine resources for their execution (cf. Example 2.7). In the former case, the activity is allocated to process participants during process execution. In the latter case, the activity is handled by the PAIS in an automated way by directly invoking an associated application service without requiring any user interaction.

Example 2.7 (Atomic Activity). Consider the prespecified process model from Fig. 2.2. *Perform Checkup* constitutes an atomic activity performed by a user with the role *Physician*, whereas the activity *Provide Postsurgical Care* has to be performed by a user with the role *Nurse*.

Activity *Print Report* from Fig. 2.1, in turn, constitutes an example of an atomic activity that can be handled in an automated way.

A *complex activity*, in turn, represents a step in a process model referring to a *subprocess model*. This signifies that the subprocess model implements the activity. Every time a complex activity gets enabled during process execution, its corresponding subprocess model is executed. Generally, a subprocess is described by its own process model and may have both input and output data containers to pass data between it and the subordinate process. Basically, subprocesses constitute a powerful concept for describing the common parts of different process models and so for enabling their reuse. Furthermore, the use of complex activities allows process designers to hierarchically structure the overall process (e.g., limit the number of activities contained in a (sub)process model).

Example 2.8 (Complex Activity). Consider again the prespecified process model from Fig. 2.2: *Perform Surgery* constitutes a complex activity which refers to a subprocess that aggregates a number of related activities not depicted in Fig. 2.2 (e.g., *Prepare Surgical Suite*, *Check Patient Record*, and *Perform Surgical Intervention*).

2.4.2 Behavior Perspective

The *behavior perspective* captures the *dynamic behavior* of an executable process model. For example, in activity-centric process models this corresponds to the *control flow* between the process activities. A control flow schema includes information about the order of the activities or the constraints for their execution. As we will see in later chapters of this book, depending on the process modeling language used and its underlying paradigm (prespecified, loosely specified, or data-driven process models), major differences can exist regarding the specification of the behavior perspective.

Example 2.9 (Behavior Perspective: Prespecified Process Models). Consider Fig. 2.2. Among others, it shows the behavior perspective of a prespecified process model. First, the depicted model contains activity sequences; e.g., activity *Admit Patient* (10) is followed by *Prepare Patient* (11), which, in turn, is followed by *Send Patient to Surgical Suite* (12). Second, the process model contains parallel splits of the control flow; e.g., *Create Surgery Report* (16) and *Order Lab Test* (17) may be performed in parallel. As another example, consider the process model from Fig. 2.1 that contains conditional branchings; e.g., either activity *Schedule X-rays* or *Reject Order* may be executed. Furthermore, this process model contains a loop structure embedding the two activities *Create Report* and *Check Validity*; i.e., these two activities may be repeated if a revision is required.

Existing languages for defining prespecified process models offer a variety of control flow elements [142], e.g., for defining sequential, conditional, parallel, and iterative activity executions. Chapter 4 presents these and other *control flow patterns* for prespecified process models and discusses how they can enable *flexibility-by-design*.

Prespecified process models prescribe how, and in which order, the activities of a process have to be executed. Opposed to this, the behavior of a constraint-based process model allows for loosely specified processes, and rather describes *what* shall be done by defining a set of activities and a set of constraints prohibiting undesired behavior [19, 243].

Example 2.10 (Behavior Perspective: Loosely specified Process Models). The constraint-based process model from Fig. 2.3 comprises activities *Examine Patient*, *Prescribe Medication*, *Perform X-rays*, *Prescribe Sling*, *Prescribe Fixation*, *Perform Surgery*, *Apply Cast*, and *Prescribe Rehabilitation*. Moreover, this process model comprises several constraints prohibiting undesired execution behavior. For example, each patient gets at least one out of four treatments (i.e., *Prescribe Sling*, *Prescribe Fixation*, *Perform Surgery*, or *Apply Cast*). Furthermore, activities *Apply Cast* and *Prescribe Fixation* are mutually exclusive. Finally, activity *Perform X-rays* needs to be executed before any treatment (except *Prescribe Sling*) may take place.

Note that Example 2.10 refers to a constraint-based process model. Generally, loosely specified processes may be partly prespecified, but contain unspecified parts which are detailed by end-users during process execution (cf. Chap. 11).

As discussed in the context of Example 2.6, there are processes which cannot be straightjacketed into activities. Instead, a tight integration of process and data is needed. In accordance to a data model comprising object types and object relations, the modeling and execution of processes can be based on two levels of granularity: *object behavior* and *object interactions*. First, *object behavior* determines in which order and by whom object attributes may be read or written, and what the valid attribute settings are. To achieve this end, a set of states may be defined for each data object type, each of them postulating the specific object attribute values to be set. More precisely, a state can be expressed in terms of a data condition referring to a number of attributes of the object type. The second level of process granularity concerns the *object interactions* between the instances of the same or of different object types. Note that whether the processing of a particular object instance may proceed also depends on the states of other object instances processed in parallel.

Example 2.11 (Behavior Perspective: Data-Driven Processes). We refer to the recruitment process from Example 2.6.

Object Behavior. Consider object type *Review* and its attributes (cf. Fig. 2.7a), the abstract states of this object type (cf. Fig. 2.7b), and its behavior expressed in terms of a state diagram (cf. Fig. 2.7c). The latter restricts possible state transitions; i.e., for each reachable state, possible successor states are defined. More precisely, a *review* must be first initiated by a *personnel officer*. The employee may then either decline or accept performing the *review*. In the latter case, he submits the review back to the personnel officer. Furthermore, states are linked to object attributes; i.e., once a state is entered, it may only be left if certain attribute values are set. For example, state *initiated* may only be left if values are assigned to attributes *applications* and *employee*.

Object Interactions. The behavior perspective of data-driven processes not only deals with the behavior of single objects, but also considers the interactions between them. Consider the scenario from Fig. 2.4. Assume that a *personnel officer* announces a *job*. Following this, *applicants* may submit their *applications*. After receiving an *application*, the *personnel officer* requests internal *reviews* for it. If an *employee* acting as referee proposes to invite the *applicant* to come in, the *personnel officer* will conduct an *interview*. Based on the results of *reviews* and *interviews*, the *personnel officer* makes a decision about the *application*. Finally, in the case of acceptance the *applicant* is hired. Obviously, whether one may continue with the processing of a particular object depends on the states of other objects as well (see Chap. 14 for details).

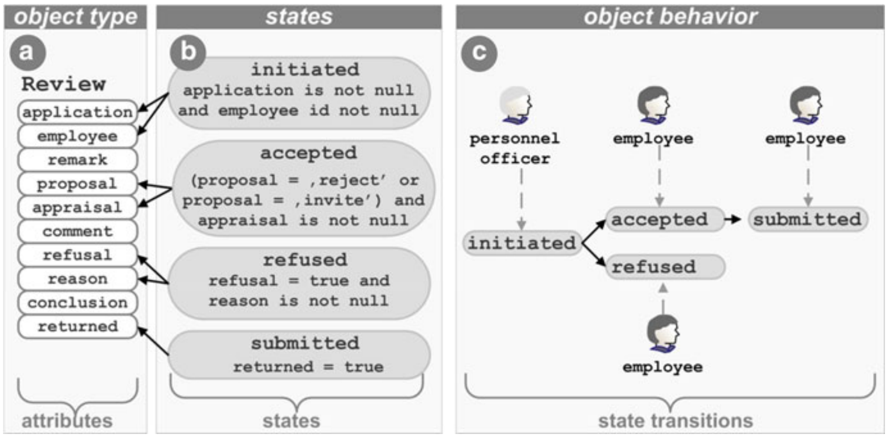

Fig. 2.7 Object behavior

2.4.3 Information Perspective

In activity-centric process models (i.e., prespecified or constraint-based process models), the *information perspective* usually comprises a set of *data objects* as well as the *data flow* between the activities. The latter describes which activities may read or write which data objects (including data mappings and data type conversions where required) [310]. Generally, in activity-centric PAISs, a distinction is made between *application data*, *process-relevant data*, and *process control data* [367]. *Application data* is specific to a particular application and is usually managed by the application services invoked in the context of an activity execution (i.e., it is outside the control of the PAIS). *Process-relevant data*, in turn, is used by the PAIS to evaluate state transitions within process instances, i.e., to decide which execution paths shall next be taken at respective split nodes. Finally, *process control data* comprises information about the current state of a process as well as its execution history.

Example 2.12 (Information Perspective: Activity-centric Processes). Consider the prespecified process model from Fig. 2.1. Its information perspective includes data objects (and exchanged messages respectively) like *Medical Order* and *Medical Report*. Usually, these objects are stored as application data in an application system, while the generic process support services of the PAIS only refer to them via object identifiers. An example of process-relevant data needed for evaluating transition conditions is the *order status* indicating whether an order is accepted. Finally, when executing a process instance, respective control data (e.g., the time at which activities are started or completed) is logged.

In data-driven process models, in turn, the information perspective is tightly integrated with the behavior perspective. Typically, the information perspective captures object types, their attributes, and their interrelations [156, 158, 225], which together form a data structure (cf. Fig. 2.4). At run-time the different object types comprise a varying number of interrelated object instances, whereby the concrete instance number may be restricted by lower and upper bounds. Typically, object instances of the same object type differ in their attribute values as well as their interrelations.

Example 2.13 (Information Perspective: Data-driven Processes). An example illustrating the information perspective of a data-driven process is depicted in Fig. 2.4 and has been explained in detail in the context of Example 2.6. For instance, for a particular *application* only two *reviews* might be requested, while for others three *reviews* are initiated.

2.4.4 Organization Perspective

The *organization perspective* deals with the assignment of human activities to organizational resources. To enable such an assignment, a process model usually contains references to an organizational model [298]. An organizational model, in turn, captures entities like *actors*, *roles*, or *organizational units* as well as their relationships (e.g., *is-manager* or *is sub-ordinate*). Furthermore, it may incorporate a variety of attributes associated with those entities (e.g., skill or role). Typically, an organizational model incorporates concepts such as hierarchy, authority, and substitution as well as attributes associated with organizational roles. The latter refer to a group of actors exhibiting a specific set of attributes, qualifications, or skills. Simply speaking, any actor having the role required by a particular activity may perform this activity. Example 2.14 illustrates the organization perspective.

Example 2.14 (Organization Perspective: Activity-centric Processes). Consider the process depicted in Fig. 2.2. It involves different organizational units (i.e., outpatient department, surgical ward, and surgical suite) as well as different user groups. For example, activities *Perform Checkup* and *Examine Patient* need to be carried out by a staff member of the *Outpatient Department* and possessing the role *Physician*. Further, there might be additional constraints concerning the execution of single activities; e.g., it might be required that the same physician who performs the medical checkup should also examine the patient; i.e., the concrete assignment of an actor to an activity at run-time may depend on the process execution history. Generally, information on user roles and organizational units is captured in the organizational model of the hospital.

Regarding data-driven processes, forms are usually used to enable access to selected attributes of an object instance (cf. Example 2.15).

Example 2.15 (Organization Perspective: Data-driven Process Models). Consider the scenario from Example 2.6. Here, a *review* form has to be filled by *employees* from various *company divisions*. Furthermore, undesired updates of these attribute values have to be prevented after reaching certain states; e.g., after an *employee* from a company division has submitted her *review*, she is no longer allowed to change the value of attribute *recommendation*.

2.4.5 Operation Perspective

The *operation perspective* of a PAIS that is based on activity-centric process models (i.e., prespecified or constraint-based process models) covers the implementation of the process activities; i.e., the business functions to be performed when executing these activities. Many PAISs treat activity implementations as a *black-box*; i.e., they coordinate the sequence of activities independent from their implementation [178]. More precisely, the application service invoked in the context of an activity expects that its input parameters are provided upon invocation by the run-time environment of the PAIS; then the service only has to take care that correct values for its output parameters are provided (cf. Fig. 2.8).

The application service linked to a particular activity may either be implemented from scratch or be reused from a service repository. Generally, a PAIS should be able to invoke arbitrary application services (e.g., Web Services, EJB Components, Office Applications) in the context of an activity execution. This heterogeneity, in turn, should be hidden from application developers, which necessitates appropriate service abstraction as well as service invocation mechanisms. As examples, consider an *Enterprise Service Bus* for invoking web services or an *Application Server* supporting EJB components.

Regarding Examples 2.2 and 2.3, most human activities can be implemented in terms of electronic forms. In addition, several connectors for integrating legacy applications (e.g., to document a surgery or access the electronic patient records in a

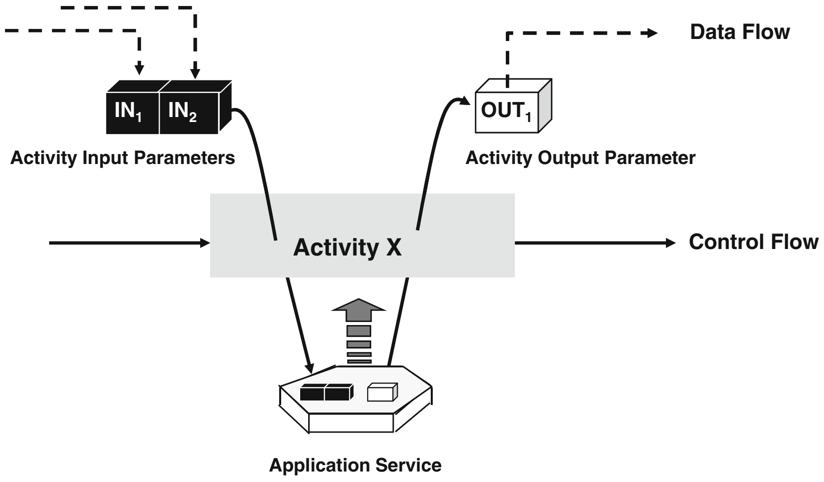

Fig. 2.8 Passing data between a PAIS and an invoked application service

hospital information system) are required. Generally, the IT support of prespecified processes is related to *enterprise application integration* [203] as illustrated by Example 2.16 (for more details on this topic we refer to [34]).

Example 2.16 (Hospital Application Integration). The architecture of a typical hospital information system is characterized by many different departmental systems that have usually been optimized for the support of different categories of business functions (i.e., services provided by different medical disciplines like radiology, cardiology, or pathology), but not for the support of cross-departmental business processes. The need to consolidate the data produced by these ancillary systems with a global (patient-centered) view and to support the cross-departmental processes has been a prime mover in the development of standards for data and message interchange in health care, as well as the enactment of similar standards in other application domains. These standards also play an important role, not only when cross-departmental processes are supported, but cross-organizational ones as well. Today, HL7 is the leading standard for systems integration in health care and may be also used to integrate health care application services with clinical PAISs [172].

In data-driven process models, the operation perspective is usually represented by *user forms* [24, 158]. When executing an electronic form related to a data-driven process, attributes of the corresponding object instance may be read, written, or updated using the respective form (e.g., the form an applicant may use for entering his application data). Generally, forms provide *input fields* (e.g., text-fields or check-boxes) for writing and *data fields* for reading selected attribute values of object instances, depending upon the object state and the process state.

2.4.6 Time Perspective

The *time perspective* captures temporal constraints that need to be obeyed during process execution. For example, consider an *activity deadline*; i.e., a time-based scheduling constraint requiring that a particular activity has to be completed by a certain date (i.e., the *activity deadline*). Typically, such a deadline is dynamically set during process execution. Generally, a PAIS should take care that activity deadlines are met or—if a deadline expires—that appropriate *escalation* and *notification* procedures are triggered (cf. Chap. 6); e.g., reminding a process participant to work on a particular activity or informing a process owner about the expiration of the deadline [21].

Note that in actual practice, many other temporal constraints exist (see [170] for a representative collection). For example, the *time patterns* described in [170] allow specifying activity durations as well as time lags between activities or—more generally—between process events (e.g. milestones). Furthermore, there exist patterns for expressing temporal constraints in connection with recurrent activities (e.g., cyclic flows and periodicity). Since the focus of this book is not on the time perspective, we will omit further details and instead refer interested readers to [170].

Example 2.17 (Time Perspective).

- In the context of Examples 2.2 and 2.3, a number of existing temporal constraints have not yet been touched upon; e.g., a surgery is usually scheduled for a particular date. Furthermore, a patient record has to be checked at least 1 day before the scheduled surgery takes place (minimum time distance). Finally, a checkup of a patient should not take longer than 30 min (activity duration).
- Regarding the data-driven process illustrated in Example 2.6, applications for a particular job vacancy have to be submitted by a certain date (deadline). Furthermore, a requested review has yet to be performed by an employee from a company division within the specified time frame (duration). Finally, job interviews must take place on a fixed date.

2.5 Components of a PAIS

This section summarizes basic components and artifacts of a PAIS.

2.5.1 Overview

As depicted in Fig. 2.9, a distinction is made between the *process type* and *process instance level*. While the former defines the schemes for executable process models, the latter refers to the execution of related process instances, i.e., single enactments of an executable process model referring to particular business cases. Accordingly, a PAIS distinguishes between build- and run-time components. While *build-time* components enable the creation and management of the type-level artifacts of the PAIS, *run-time* components refer to the process instance level and support the creation, execution, and management of process instances.

Figure 2.10 shows an overview of the build- and run-time environment of a PAIS. Regardless of the paradigm chosen (i.e., prespecified, constraint-based, or data-driven process), the *build-time environment* includes tools for defining, configuring, and verifying executable process models. The core of the *run-time* environment, in turn, is built by a *process engine*. The latter is a generic software service that allows creating, executing, and managing the instances of executable process models. This includes the creation of new process instances, the run-time interpretation of these process models according to the defined behavior, the execution of activity instances (including sub-process instances), the management of user worklists, and the invocation of application services (e.g., web services or user forms) in the context of activity executions. Furthermore, the run-time environment comprises *end-user tools* enabling access to worklists or status monitors.

2.5.2 Build-Time Environment

Before implementing a business process in a PAIS, it must first be decided which parts of the process are to be automated. Then, an executable process model covering these parts needs to be created. Usually, the latter task is accomplished using the *process model editor* of the PAIS (cf. Fig. 2.11). This build-time component allows process designers to define, configure, and verify the different perspectives of an executable process model. The latter includes the activities of the executable process model (i.e., function perspective) as well as the control and data flow between them (i.e., behavior and information perspectives). Furthermore, for each atomic activity an application service (e.g., a user form or web service) has to be provided either through implementation or reuse from a service repository (i.e., operation perspective). In turn, for complex activities, an executable subprocess model must be defined. Finally, for human activities, so-called actor expressions have to be

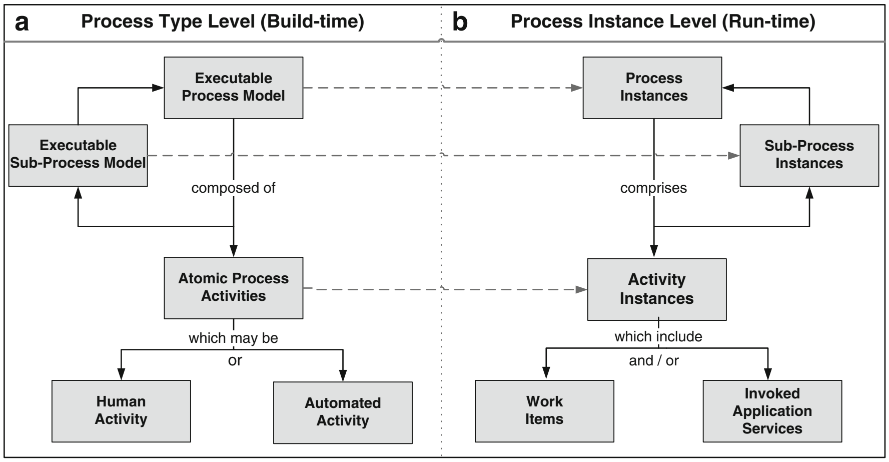

Fig. 2.9 Build-time and run-time artifacts of a PAIS (adopted from [367])

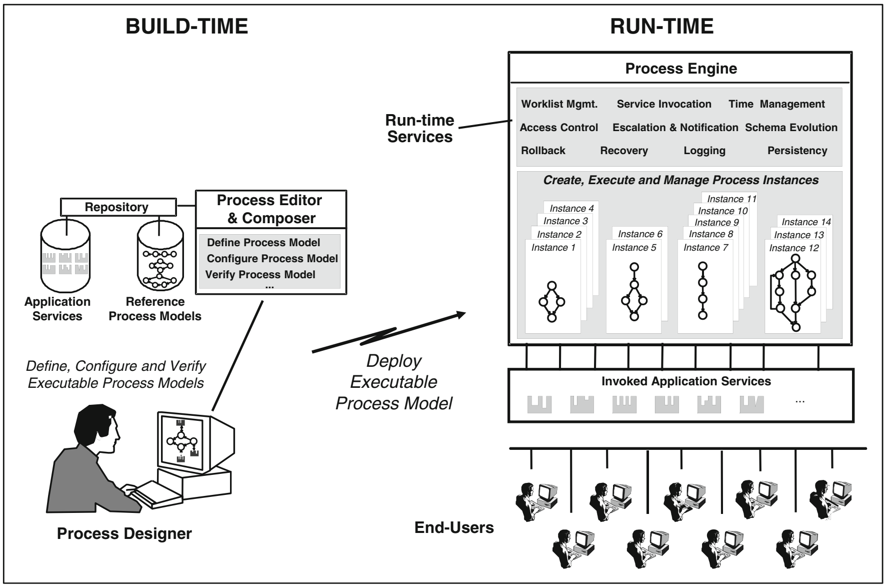

Fig. 2.10 Build-time and run-time environment of a PAIS

specified to enable the PAIS to assign these activities to potential actors during run-time (i.e., organization perspective). In turn, an actor expression refers to entities from the *organizational model* of the PAIS, such as roles or organizational units (cf. Fig. 2.11).

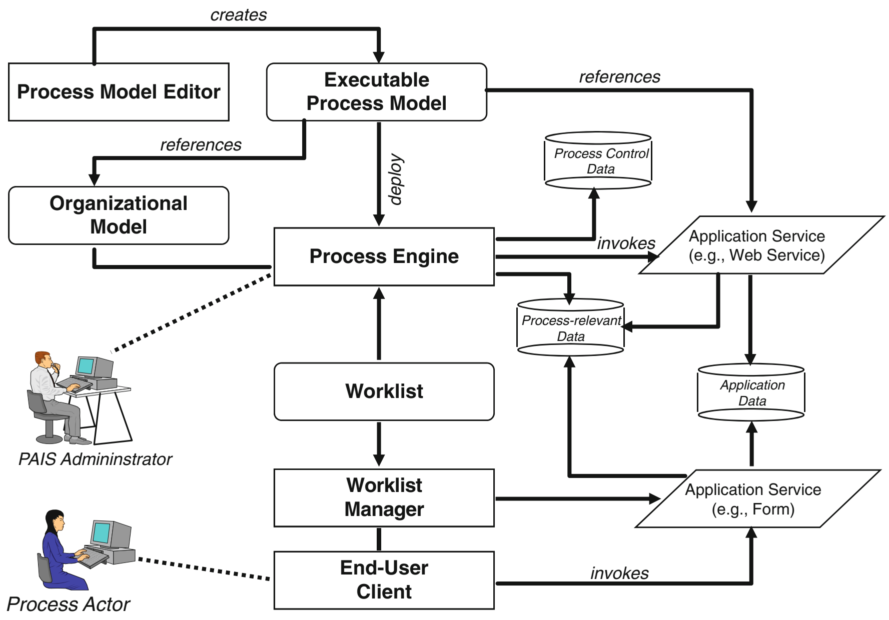

Fig. 2.11 Basic components of a PAIS (adopted from [367])

Regardless of the process modeling approach used (i.e., prespecified, constraint-based, or data-driven process), the build-time components should enable comprehensive checks of executable process models in order to exclude any error or undesired behavior during run-time. Respective verification tasks concern all process perspectives, e.g.:

- *Function perspective*: Do activity labels comply with existing guidelines and taxonomies [206]? Are required activity attributes completely specified?
- *Behavior perspective*: Will the instances of an executable process model always complete properly or may undefined states (e.g., deadlocks) occur during run-time? Are there activities that will never be executed?
- *Information perspective*: Does each data object have a defined data type? Will there be missing data or unnecessary data during process execution?
- *Organization perspective*: Do all organizational entities to which actor expressions of human activities refer exist in the organizational model? Can we ensure that there will be always at least one user authorized to execute a particular human activity?
- *Operation perspective*: Does each atomic activity have an assigned application service? Will there always be assigned values for the input parameters of an invoked application service at run-time?

Focusing on the behavior and information perspectives, later chapters of this book will show how modeling and verification tasks look like for prespecified, constraint-based, and data-driven process models.

2.5.3 *Run-Time Environment*

After releasing an executable process model, it can be deployed to the PAIS run-time environment whose core component is the *process engine*. A process engine allows creating, executing, and managing process instances related to the same or to different process models; i.e., a process engine constitutes a software service providing the run-time environment for executing a collection of process instances.

Figure 2.11 shows build-time and run-time components of a PAIS as well as their relationships. Core services provided by these PAIS components are as follows:

- Creating new instances of an executable process model.
- Executing process instances and related activities through interpretation of executable process models.
- Managing process instance data (i.e., control data and process-relevant data).
- Creating work items for instances of human activities and assigning these work items to the worklists of qualified actors.
- Orchestrating the application services and subprocesses linked to the activities of a process model according to the defined process logic.
- Invoking the right application service when starting the execution of an activity and exchanging parameter data with this service.
- Monitoring the progress of process instances and logging relevant execution events to ensure process traceability.

In the following, the notions of process instance, activity instance, and work item will be explained in detail.

2.5.3.1 Process Instances and Their Life Cycle

Once an executable process model has been deployed to a process engine, new *process instances* can be created and executed according to this model. Generally, several instances of the same process model may exist representing different *business cases* (e.g., treatments of different patients). The process engine employs a *state model* to control the concurrent execution of these process instances; i.e., each process instance exhibits an internal state representing its progress toward completion and its status with respect to its activities and data objects.

Process Instance Life Cycle. Figure 2.12 depicts the life cycle of a process instance. A newly created process instance has state `Created`. When starting its execution, this state changes to `Running`; i.e., the process model is then interpreted by a process engine, and activities whose preconditions are met become

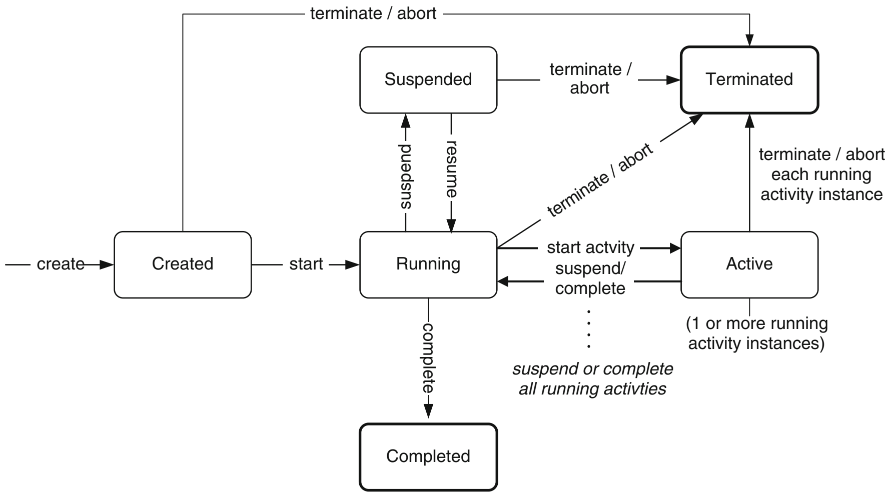

Fig. 2.12 Possible states and state transitions of a process instance

enabled. As soon as at least one of the enabled activities is running (i.e., it has been started), the process instance enters state `Active`. The distinction between the two states `Running` and `Active` is useful since different actions may be applied in these states. A process instance without currently running activities can be easily suspended or completed. Conversely, this is not immediately possible with a process instance with running activity instances. Furthermore, in the case a process instance is abnormally terminated or aborted (i.e., the instance enters state `Terminated`), different actions are required, depending on the concrete state of the process instance; e.g., for a process instance in state `Active`, each running activity instance needs to be terminated before the process instance itself may enter state `Terminated`.

Example 2.18 (Process Instance). Figure 2.13 shows an example of a process instance representing a *Request for Credit* by one particular customer. In detail, activities *Collect Credit Data* and *Assess Risk* have been completed, while activity *Accept Credit* is enabled. Activities *Request Approval* and *Refuse Credit*, in turn, have been skipped during the execution of the process instance. Furthermore, the instance state includes information about produced and consumed data objects, as well as about the actors who have worked on human activities (not depicted in Fig. 2.13).

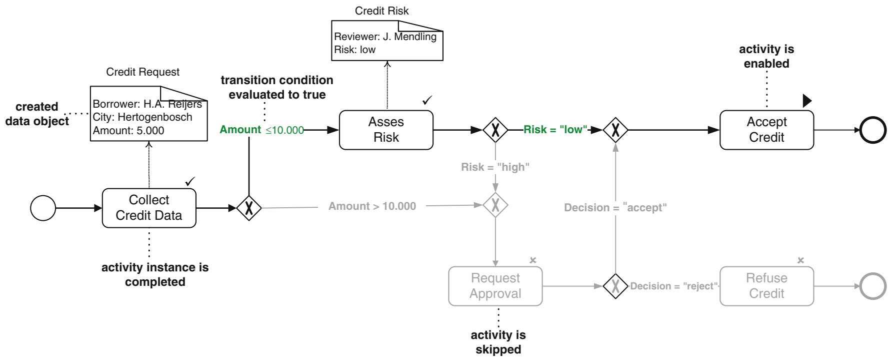

Fig. 2.13 Internal state of a process instance

Event Logs. Generally, all relevant events occurring during the execution of a process instance (e.g., start and completion of activities) are recorded in a an *event log* (also denoted as an *execution log* or *audit trail*). Respective event logs provide detailed information about the actual sequences of activities, key attributes of the performed activity instances (e.g., start and completion times), and the resources or human actors that performed these activities. Based on event logs, the PAIS additionally offers a run-time component for *monitoring* and *visualizing* the progress of its running process instances [53]. In particular, process monitoring relieves staff members from manually keeping track of their processes.

Example 2.19 (Process Monitoring). Regarding the process dealing with order entry and result reporting from Example 2.2, a process monitoring component will allow hospital staff to answer questions like “Has an ordered X-ray already been made?” or “Why is there no medical report for a previously ordered X-ray?”.

Note that in connection with data-driven processes, respective monitoring components usually do not only provide a *process-oriented view*, but also a *data-oriented view* for accessing business data at any point in time.

2.5.3.2 Activity Instances and Their Life Cycle

When the preconditions for executing a particular activity are met during run-time, a new instance of this activity is created. Hence, an *activity instance* represents a single invocation of an activity during the execution of a particular process instance. Furthermore, an activity instance utilizes data associated with its corresponding process instance and itself produces data utilized by succeeding activities.

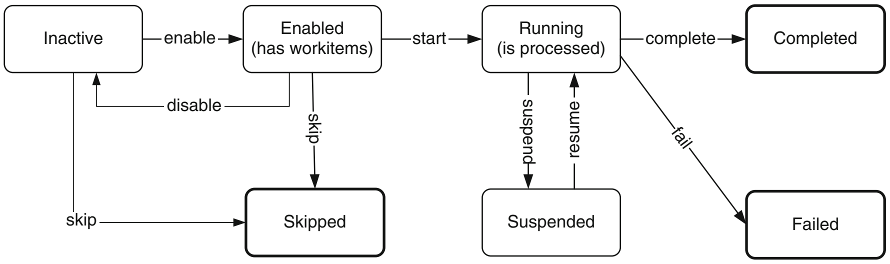

Fig. 2.14 States and state transitions of an activity instance

Activity Instance Life Cycle. Figure 2.14 illustrates the life cycle of a single activity instance. When the preconditions for executing an activity are met during process execution, the state of the activity instance changes from `Inactive` to `Enabled`. If no human interaction is required, the activity instance immediately enters state `Running` and its corresponding application service is invoked. In this context, process-relevant data is passed from the process engine to the input parameters of the invoked application service, which then processes these data (cf. Fig. 2.15). When completing the execution of the application service, in turn, its output parameters are mapped to process-relevant data, which may then be accessed in subsequent activity executions of the same process instance. Otherwise (i.e., human interaction is required), corresponding *work items* are created for qualified actors and added to their worklists. As soon as one of these actors starts processing this work item, the activity instance switches to state `Running` (cf. Sect. 2.5.3.3). Finally, when an activity instance completes, its state changes to `Completed`. Usually, this is followed by an evaluation of the preconditions of subsequent activities.

To cover more advanced scenarios, three states have to be added as indicated in Fig. 2.14. First, an activity instance in state `Inactive` or `Enabled` may be skipped (i.e., it enters state `Skipped`) if an alternative path is chosen for execution. Second, a human activity (e.g., writing a medical report) in state `Running` may be suspended (i.e., it enters state `Suspended`) and resumed later (i.e., it reenters state `Running`). Finally, if a running activity instance fails due to technical or semantic errors, it then switches to state `Failed`.

2.5.3.3 Worklists and Work Items

When a human activity becomes enabled during the execution of a process instance, the PAIS first determines all actors qualifying for this *activity instance*. Basic to this is the *actor expression* that is associated with the respective activity and can be used for querying the organizational model of the PAIS.

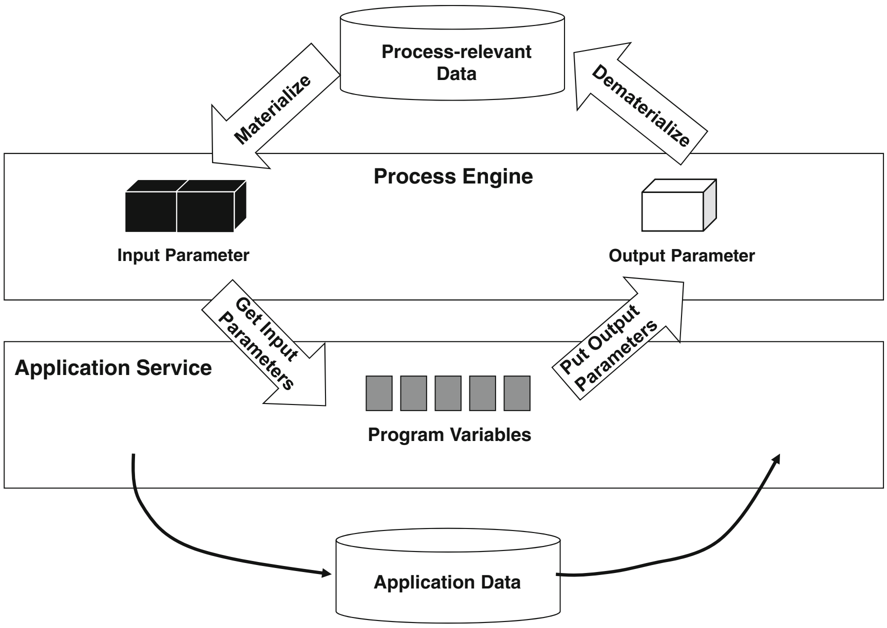

Fig. 2.15 Exchanging data between process engine and invoked application service

Example 2.20 (Actor Assignment for a Human Activity). Consider the process model from Fig. 2.2. Activity *Perform Checkup* will be offered to all actors with role *Physician* and who are also a member of the *outpatient department*.

For each potential actor, a *work item* referring to the activity instance is created and added to his *worklist*, i.e., work items related to a particular activity instance may be added to different user worklists. Generally, a worklist comprises all work items currently offered to, or processed by, a user. Example 2.21 illustrates the relationships that may exist between executable process models, process instances, and work items. Note that this example abstracts from those actor expressions that have led to the creation of respective work items.

Example 2.21 (Worklists and Work Items). Consider Fig. 2.16, which depicts two prespecified process models (i.e., *Biopsy* and *Medication*) and five related process instances I_1 to I_5. While instances I_1, I_2, and I_3 are running on process model *Biopsy*, instances I_4 and I_5 are based on process model

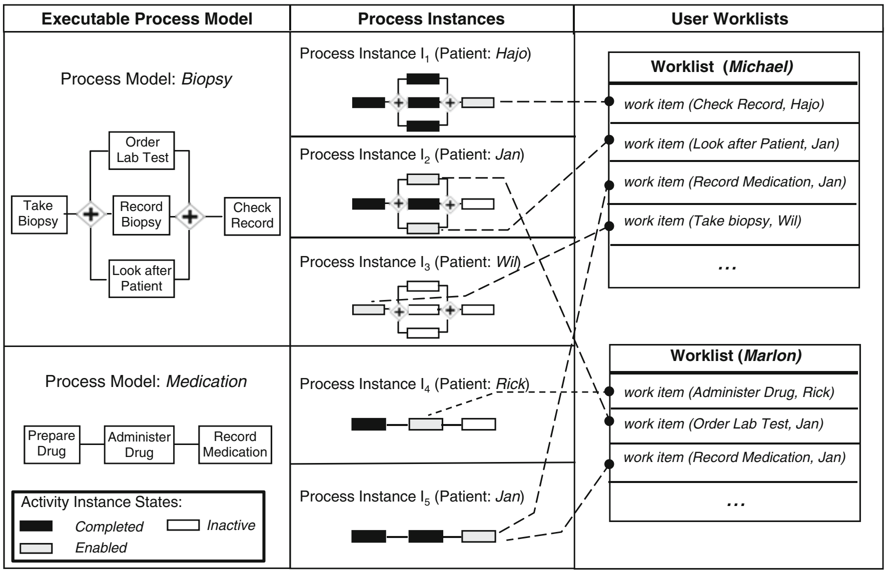
Executable Process Model
Process Instances
User Worklists
Process Model: Biopsy
Take Biopsy
Order Lab Test
Record Biopsy
Look after Patient
Check Record
Process Model: Medication
Prepare Drug
Administer Drug
Record Medication
Activity Instance States:
Completed
Inactive
Enabled
Process Instance I1 (Patient: Hajo)
Process Instance I2 (Patient: Jan)
Process Instance I3 (Patient: Wil)
Process Instance I4 (Patient: Rick)
Process Instance I5 (Patient: Jan)
Worklist (Michael)
work item (Check Record, Hajo)
work item (Look after Patient, Jan)
work item (Record Medication, Jan)
work item (Take biopsy, Wil)
...
Worklist (Marlon)
work item (Administer Drug, Rick)
work item (Order Lab Test, Jan)
work item (Record Medication, Jan)
...

Fig. 2.16 Executable process models, process instances, and work items

Medication. Furthermore, the depicted worklists of users *Michael* and *Marlon* comprise work items relating to process instances I_1 to I_5.

- An instance of activity *Check Record* belonging to I_1 is enabled and a corresponding work item has been added to the worklist of user *Michael*.
- Regarding I_2, two work items exist belonging to different activity instances. One of them refers to an instance of activity *Order Lab Test*—a corresponding work item has been added to the worklist of user *Marlon*. The other one refers to an instance of activity *Look after Patient*. For this activity, a corresponding work item is added to the worklist of user *Michael*.
- For activity instance *Record Medication* of I_5 there are two work items assigned to the worklists of *Michael* and *Marlon*.

Generally, process participants interact with a PAIS via end-user clients and the worklists displayed by them. When an actor allocates a work item from his worklist, all work items related to the same activity instance are removed from the worklists of other users. Further, as illustrated by Fig. 2.17, the user to whom the work item is allocated may then trigger the start of the application service associated with the corresponding activity instance.

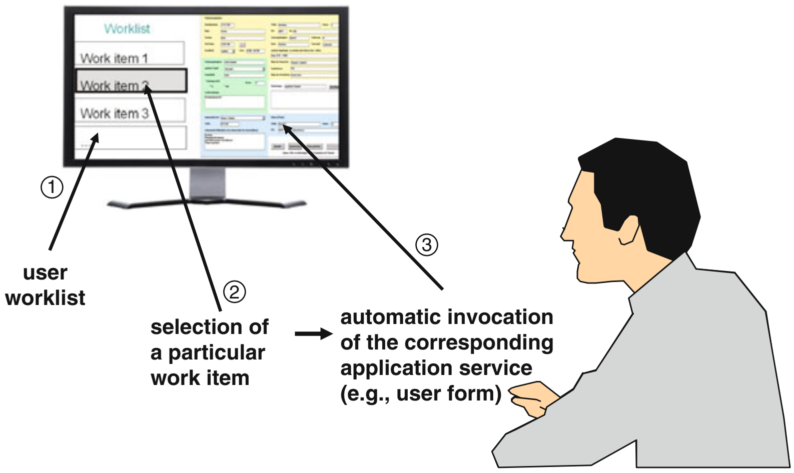
Worklist
Work item 1
Work item 2
Work item 3
①
user worklist
②
selection of a particular work item
③
automatic invocation of the corresponding application service (e.g., user form)

Fig. 2.17 Selecting a work item from a worklist

Work Item Life Cycle. As illustrated by Fig. 2.16, several work items may exist for a particular activity instance. In the following, the states of a single work item and its processing are discussed.

Figure 2.18 illustrates the *work item life cycle* from the perspective of one particular actor to whom this work item is assigned [311]: A work item typically progresses from state `Offered` to `Allocated` to `Started`, and finally to `Completed`. Initially, the work item has state `Offered`; i.e., it is offered to all qualified actors (e.g., all user possessing a particular role). If one of these actors wants to perform the task associated with the work item, he needs to issue an allocate request. The work item is then allocated to this actor. At the same time, all work items from other worklists referencing the same activity instance are removed. When the actor who allocated the work item wants to start its execution, he issues a start request, and the state of the work item changes to `Started`. Finally, once the work item is processed, the actor issues a complete request and the state of the work item changes to `Completed`. Three additional states need to be added to this life cycle as indicated by the dotted arcs in Fig. 2.18. First, a work item will immediately change from its initial state `Offered` to state `Withdrawn` if another work item belonging to the same activity instance, but being offered to a different actor, is allocated by that actor. Second, the processing of a started work item may be temporarily suspended (i.e., the state of the work item switches from `Started` to `Suspended`) and later be resumed. Third, if the execution of a work item fails, its state will change to `Failed`. The latter corresponds to a termination action in relation to the work item which is outside the control of the actor. We will extend the life cycle from Fig. 2.18 in Chap. 6 to show what additional support is needed to flexibly cope with exceptional situations during run-time.

In principle, process participants do not need to know the exact logic of a process instance when working on a corresponding work item; i.e., they may solely interact

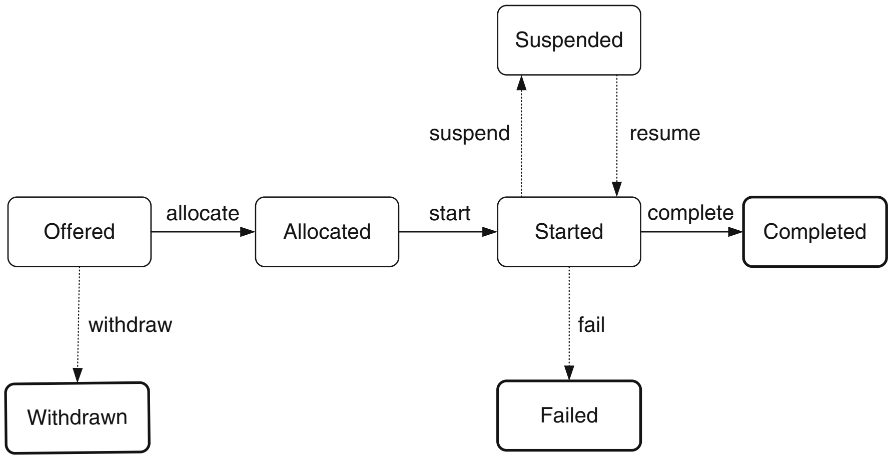

Fig. 2.18 Work item life cycle from the perspective of a particular actor

with the PAIS via their worklist (cf. Fig. 2.17). When completing the processing of a work item, new work items referring to subsequent activity instances are created and added to user worklists.

2.6 Summary

Turning away from hard-coded process logic toward explicitly specified process models significantly eases PAIS development and maintenance. In summary, a PAIS

- knows the logic of the supported processes; i.e., processes are explicitly described in terms of executable process models (e.g., comprising a set of activities and a number of constraints for their execution).
- ensures that activities are executed in the specified order or considering the specified constraints (i.e., the PAIS manages the flow of control during run-time).
- controls the flow of data between the activities; i.e., the output data of a particular activity can be consumed as input data by subsequent activities.
- knows the application service to be invoked when an atomic activity is started.
- assigns work items related to human activities to the worklists of authorized users and manages these worklists.
- provides reminders if users do not complete an activity instance before a certain deadline is reached.
- enables end-users to monitor the progress of process instances and to trace their previous execution.

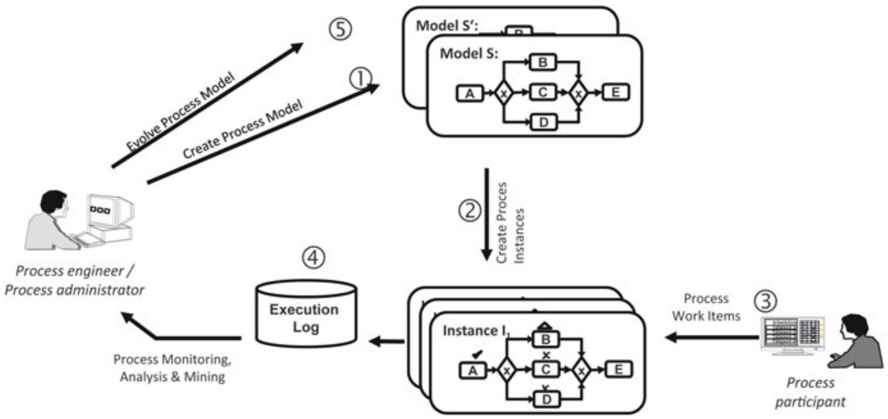

Fig. 2.19 Traditional process life cycle

Traditional PAISs enable process life cycle support as depicted in Fig. 2.19: At build-time, an initial representation of the business process to be supported is created either by explicitly modeling the process based on the result of a process analysis or by discovering the process model through the mining of execution logs [11, 133] (1). At run-time, new process instances can be created from the executable process model (2), each representing a concrete business case. In general, process instances are executed according to the process model they were originally derived from. While automated activities are immediately executed once they become enabled, nonautomated activities are assigned to the worklist of qualified actors who may perform the respective activities (3). Thereby, execution logs record information about the start and completion of activity instances as well as their chronological order (4). The analysis of respective logs by a process engineer and/or process intelligence tools allows discovering malfunctions or bottlenecks. In turn, this triggers the evolutionary change of the process model (5).

We will extend this life cycle in subsequent chapters to accommodate the different flexibility needs for PAISs as discussed in Chap. 3.

Exercises

2.1. Process Perspectives

Think of a business process you are familiar with that utilizes a PAIS (e.g., ordering goods in a Web shop or performing a financial transaction using Internet banking). Give examples of the different perspectives on a PAIS using this scenario.

2.2. Process Instances, Activity Instances and Work Items

(a) Consider the states of a process instance as illustrated in Fig. 2.12. Explain the difference between states `Completed` and `Terminated`.
(b) What is the relationship between the life cycle of an activity instance (cf. Fig. 2.14) and the one of a single work item (cf. Fig. 2.18)?
(c) Consider the life cycle of an activity instance from Fig. 2.14. What exactly is the difference between states `Enabled` and `Running`?
(d) Which of the following statements are true?

- A worklist may contain more than one work item.
- Each activity instance is associated with exactly one work item.
- For a process instance in state `Running` multiple work items related to the same activity instance may exist at a certain point in time.
- A work item switches from state `Offered` to state `Withdrawn` if another work item related to the same activity instance is allocated by a user.

2.3. Application Data, Process-Relevant Data, and Process Control Data

(a) Describe the differences between application data, process-relevant data, and process control data. Give examples.
(b) Explain how data may be exchanged between a started activity instance and the application service invoked during its execution.
(c) Describe how data is passed between the activities of a process instance.

2.4. Build-Time and Run-Time Components of a PAIS

Give examples of build- and run-time components of a PAIS. What services are offered by them?