1 Information Systems: Introduction and Concepts

Information systems have become the backbone of most organizations. Banks could not process payments, governments could not collect taxes, hospitals could not treat patients, and supermarkets could not stock their shelves without the support of information systems. In almost every sector—education, finance, government, health care, manufacturing, and businesses large and small—information systems play a prominent role. Every day work, communication, information gathering, and decision making all rely on information technology (IT). When we visit a travel agency to book a trip, a collection of interconnected information systems is used for checking the availability of flights and hotels and for booking them. When we make an electronic payment, we interact with the bank's information system rather than with personnel of the bank. Modern supermarkets use IT to track the stock based on incoming shipments and the sales that are recorded at cash registers. Most companies and institutions rely heavily on their information systems. Organizations such as banks, online travel agencies, tax authorities, and electronic bookshops can be seen as IT companies given the central role of their information systems.

This book is about modeling business processes. A business process describes the flow of work within an organization. It is managed and supported by an information system. In this chapter, we first introduce information systems (section 1.1) and discuss different types of information systems and their roles in organizations (section 1.2). After introducing information systems, we look at the life cycle of these systems and concentrate on the important role that models play in this life cycle (section 1.3). Next, we show how to describe information systems in terms of states and state transitions (section 1.4). Although transition systems are not suitable for modeling industrial information systems and business processes, they illustrate the essence of modeling. Finally, we discuss the role of modeling and provide an outlook on the next chapters (section 1.5).

1.1 Information Systems

Organizations offer products to customers to make money. These products can be goods or services. In most organizations, huge volumes of data accumulate: data of products, data of customers, data of employees, data of the delivery of products, and data of other sources. These data therefore play an important role in contemporary organizations and must be stored, managed, and processed, which is where *information systems* come into play. Because there is no unique understanding of what an information system is, we develop a definition of an information system in this section by considering an example organization everybody should be familiar with: a family doctor.

Example 1.1 A patient who consults a family doctor usually first tells the doctor about the symptoms. With this information, the doctor examines the patient and makes a diagnosis. Afterward, the doctor determines the treatment to heal the patient. For example, based on the diagnosis, the doctor may write the patient a prescription for some medication. Finally, the doctor must document the symptoms, the diagnosis, and the treatments. Today, most doctors use a software system to record this information.

Before we provide our definition of an information system, we first explain the term "information," which can mean any of the following:

1. The *communication act* of one agent—the term "agent" may refer to any entity ranging from a person or a software component to an organization—informing another agent (e.g., by exchanging messages);
2. The *knowledge* or *beliefs* of agents as a part of their mental state; or
3. (Data) *objects* that represent knowledge or beliefs.

Example 1.2 In the example of the family doctor, the situation in which a patient informs the doctor about the symptoms is an example of a communication act. The patient and the doctor are the agents in this example. The doctor uses her knowledge and the symptoms described by a patient to examine the patient. The doctor may have beliefs about possible causes based on earlier interactions with the patient. Based on the outcomes of the examination and on prior knowledge, the doctor makes a diagnosis. The documentation of the symptoms, of the diagnosis, and of the treatments in a software system leads to the creation of data objects. These data objects represent the new knowledge and may be used for various purposes—for example, for billing the insurance company of the patient.

There are textbooks in which the authors distinguish between data, information, and knowledge. In these textbooks, the term "data" refers to the syntax, "information" refers to the interpretation, and “knowledge” refers to the way information is used. The data element “29-01-1966,” for example, may be seen as a string; in a particular context it may, however, be interpreted as the birthdate of a person, and people may use this information to congratulate this person on the twenty-ninth of January each year. In this chapter, we use the term “information” in a broader sense, as described earlier.

Having explained “information,” we can define the term “information system.” The standard definition is that *an information system manages and processes information*. This definition is general and allows different interpretations. For example, it is not clear whether “information system” refers only to software systems or also to humans, such as a family doctor who manages and processes information. For this reason, we develop a more refined definition.

The reason for “information system” having several meanings becomes clear when we consider Alter’s framework for information systems (Alter 2002) in figure 1.1. It shows an integrated view of an information system encompassing six entities: customers, products (and services), business processes, participants, information, and technology. Customers are the actors that interact with the information system through the exchange of products or services. These products are being manufactured or assembled in business processes that use participants, information, and technology. Participants are the people who do the work. Information may range from information about customers to information about products and business processes. Business processes use technology, and new technologies may enable new ways of doing work.

Customers and participants are examples of *agents*. As figure 1.1 shows, business processes play a central role in larger information systems. A business process describes the flow of work within an organization. In this book, we use the following definition of a business process adapted from work by Weske (2007).

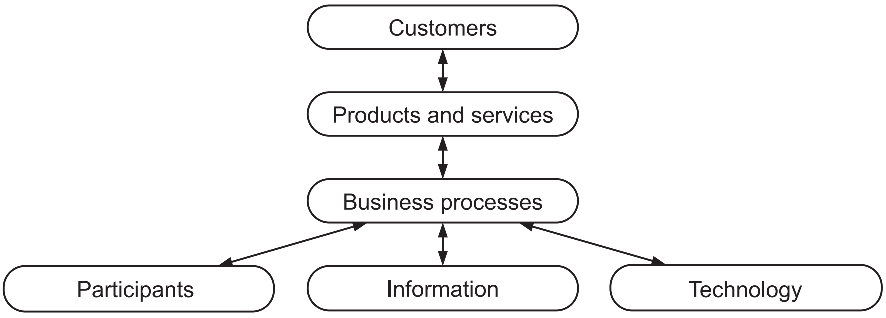

Figure 1.1
An integrated view of an information system.

Definition 1.3 (Business process) A *business process* consists of a set of activities that is performed in an organizational and technical environment. These activities are coordinated to jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.

According to this definition, a business process consists of coordinated activities. Typically, these activities must be performed in a particular order. For example, the family doctor first examines a patient and then makes a diagnosis. Although a business process is enacted by a single organization, it may interact with other business processes within and across organizational boundaries. For example, the family doctor may bill the insurance company of the patient.

Diagrams like the one in figure 1.1 illustrate why it is difficult to provide a standard definition of an information system. Some researchers and practitioners hold a view that all six elements constitute an information system; other researchers and practitioners argue that only a subset (e.g., just business processes, information, and technology) constitutes an information system.

Example 1.4 Let us pick up again the example of the family doctor. A patient serves as a customer, according to figure 1.1, and the product is health care. The business process describes the procedure of the medical treatment. It has five activities: a patient informs the doctor about the symptoms, then the doctor examines the patient, makes a diagnosis, determines the treatments, and finally the doctor enters the data into the software system. The doctor is a participant, pieces of information are the symptoms of the patient and the data added to the software system, and the doctor's software system is the technology involved.

Given these considerations, we present the following definition of an information system, which is adapted from Alter's definition (Alter 2002).

Definition 1.5 (Information system) An *information system* is a software system to capture, transmit, store, retrieve, manipulate, or display information, thereby supporting people, organizations, or other software systems.

In contrast to other definitions, we consider an information system to be a software system. A family doctor is, hence, not part of an information system. Furthermore, an information system may support not only an organization or a person but also other software systems and, hence, information systems. In addition, our definition of an information system does not require the existence of a business process; a text editor is an example of an information system that has no business process. In this book, however, we concentrate on information systems in which business processes play a central role.

Example 1.6 In the example of the family doctor, the information system is the software system that stores the data of the patient. This information system supports a person: the doctor.

1.2 Types of Information Systems

In the previous section, we defined "information system." Many types of information systems exist on the market. To illustrate this, this section first provides a broad classification of information systems. We then narrow our view to enterprise information systems and present for this class of information systems an overview of existing types of software systems. Moreover, we provide examples of typical enterprise information systems in various industries.

1.2.1 Classifying Information Systems

It is ambitious to classify the many types of information systems that have emerged in practice. Many classifications for information systems exist in the literature; see classifications by Alter (2002), Dumas, Van der Aalst, and Ter Hofstede (2005), and Olivé (2007), for instance. The problem is that classification is in flux; that is, a classification developed a few years ago is not necessarily current. As another and main limiting factor, the categories of a classification are typically not disjointed: one type of information system belongs to multiple categories. Given these problems, we present a high-level classification that distinguishes three classes of information systems.

The first class of information systems is *personal information systems*. Such an information system can manage and store information for a private person. Examples are an address book or address database and an audio CD collection.

Enterprise (or organizational) *information systems* are the second class of information systems. An enterprise information system is tailored toward the support of an organization. We distinguish between *generic* types and technologies of information systems and information systems for *certain* types of organizations. The former class of enterprise information systems supports functionality that can be used by a wide range of organizations. Examples are workflow management systems, enterprise resource planning systems, data warehouse systems, and geographic information systems. In contrast, information systems for certain types of organizations offer functionality that is tailored toward certain industries or organizations. Examples are hospital information systems, airline reservation systems, and electronic learning systems.

The third class of information systems is *public information systems*. Unlike personal information systems, public information systems can manage and store information that can be accessed by a community. Public libraries, information systems for museums, Web-based community information systems, and Web-based stock-portfolio information systems are examples of public information systems.

In this book, we concetrate on enterprise information systems. These systems play a crucial role in a wide variety of organizations and have an enormous economic value. The complexity and importance of such systems provide serious challenges for IT professionals ranging from software engineers to management consultants. Business processes and business process models play a dominant role in enterprise information systems. This explains why business process modeling is the focus of later chapters.

1.2.2 Types of Enterprise Information Systems

There are many types of enterprise information systems in practice. This section gives an overview of the most important types.

Enterprise Resource Planning Systems An *enterprise resource planning* (ERP) *system* is an information system that supports the main business processes of an organization—for example, human resource management, sales, marketing, management, financial accounting, controlling, and logistics. In the past, each business process was encapsulated in a separate information system. As most of these business processes use related data, much redundant data had to be stored within the respective information systems. The increasing number and complexity of information systems forced organizations to spend much effort in synchronizing the data of all information systems.

An ERP system is a solution to overcome these synchronization efforts by integrating different information systems. It is a software system that is built on a distributed computing platform including one or more database management systems. The computing platform serves as an infrastructure on which the individual business processes are implemented. First-generation ERP systems now run the complete back office functions of the world's largest corporations.

ERP systems run typically in a three-tier client/server architecture consisting of a user interface (or presentation) tier, an application server tier, and a database server tier. ERP systems provide multi-instance database management, configuration management, and version (or customization) management for the underlying database schema, for the user interface, and for the many application programs associated with them. As ERP systems are typically designed for multinational companies, they have to support multiple languages, multiple currencies, and country-specific business practices. The sheer size and the tremendous complexity of these software systems make them complicated to deploy and maintain.

ERP systems are large and complex software systems that integrate smaller and more focused applications; for example, most ERP systems include functionality that is also present in other enterprise information systems, such as procurement systems, manufacturing systems, sales and marketing systems, delivery systems, finance systems, and workflow management systems. We introduce these systems in the following discussion.

The market leader in the ERP market is SAP, with 43,000 customers for its system SAP ERP (data from 2009). Other important vendors are Oracle, Sage Company, and Microsoft.

Procurement Systems A *procurement system* is an information system that helps an organization automate the purchasing process. The aim of a procurement system is to acquire what is needed to keep the business processes running at minimal cost. With the available inventory, the expected arrival of ordered goods, and forecasts based on sales and production plans, the procurement system determines the requirements and generates new orders. At the same time, it tracks whether ordered goods arrive. The key point is to order the right amount of material at the right time from the right source. If the material arrives too early, money for buying the material and warehouse space to store the material will be tied up. If, in contrast, the material arrives too late, then production is disrupted. Hence, the goal is to balance reducing inventory costs with reducing the risk of out-of-stock situations.

Procurement is an important ingredient of *supply chain management* (SCM), in which coordination of the purchasing processes is not limited to two actors. Instead, SCM aims at closely coordinating an organization with its suppliers so that inefficiencies are avoided by optimizing the entire purchasing process. For example, by synchronizing the production process of an organization with its suppliers, all parties may reduce their inventories. The market leader in the SCM market is SAP with SAP SCM; competitors are Oracle and JDA Software (data from 2007).

Procurement is related to *electronic data interchange* (EDI), the electronic exchange of information based on a standard set of messages. EDI can be used to avoid delays and errors in the procurement process as a result of rekeying information. In the classical (pre-EDI) situation, a purchase order is entered into the procurement system of one organization, it is printed, and the printed purchase order is sent to the order processing department or to another organization. The information on the printed purchase order is then reentered into the procurement system. By using EDI or technology such as Web services, organizations can automate these parts of the procurement process. The purchase order is electronically sent to the processing department or to the other organization. This automation makes the overall procurement process faster and less error-prone, thereby reducing the costs for each purchase order.

Manufacturing Systems *Manufacturing systems* support the production processes in organizations. Driven by information, such as the bill of materials (BOM), inventory levels, and available capacity, they plan the production process. With increasing automation of production processes, manufacturing systems have become more and more important. For example, most steps in the production line of a car, such as welding the auto body, are performed by robots. This requires precise scheduling and material movement and, hence, a manufacturing system that supports these processes.

Material requirements planning (MRP) is an approach to translate requirements (i.e., the number of products for each period), inventory status data, and the BOM into a production plan without considering capacities. Successors, such as *manufacturing resources planning* (MRP2), also take capacity information into account. Software based on MRP and MRP2 has been the starting point for many ERP systems.

Consider an organization that produces different flavors of yogurt (e.g., strawberry, peach, and pear). The organization has several machines to produce yogurt; each machine can produce any flavor. Production planning means scheduling each machine for the flavor of yogurt it must produce. The production plan depends on the demand for each flavor and on the delivery of ingredients. Furthermore, each machine has to be cleaned at regular intervals and when the production changes to a new flavor. Calculating a production plan is a complex optimization problem, often depending on several thousand constraints. Consequently, the aim is to find a good solution rather than an optimum solution.

Sales and Marketing Systems *Sales and marketing systems* need to process customer orders by taking into account issues such as availability. These systems are driven by software addressing the four *p*'s: product, price, place, and promotion. Organizations undertake promotional activities and offer their products at competitive prices to boost sales, but a product that is not available or not at the right location cannot be sold. One prominent example of a promotional activity is a bonus card in supermarkets. Customers who register for a bonus card get a discount or a voucher. Bonus cards are an instrument for organizations to obtain personal data about their customers (e.g., age, address) and data about the buying behavior of customers (i.e., what they buy and when they buy it). These data are collected and processed by an information system. The information extracted from these data can help to improve marketing and to determine the range of products to offer.

New technologies are increasingly used to support sales over the Internet. *Electronic commerce* uses the Internet to inform (potential) customers, to execute the purchase transaction, and to deliver the product. Again, this functionality is typically embedded in an ERP system. To manage the contact with their customers, organizations use dedicated *customer relationship management* (CRM) *systems*. A CRM system has a database to store all customer-related information, such as contact details and past purchases.

This information helps tailor the marketing efforts to expected customer needs. As an example, a car dealer does not need to send information about a new expensive sports car to customers who recently bought a van or a compact car.

Delivery Systems A *delivery system* is an information system that supports the delivery of goods to customers. The task of these systems is to plan and schedule when and in what order customers receive their products. Consider, for example, a transportation company with hundreds of trucks. The planning of trips, the routing of these trucks, and reacting to on-the-fly changes require dedicated software. Creating an optimal schedule is a complex optimization problem. As circumstances—for example, traffic jams and production problems—may force rescheduling, contemporary delivery systems aim to find a good solution rather than a theoretical optimum solution. More and more delivery systems offer tracking-and-tracing functionality; for example, customers of package delivery companies, such as UPS, can track down the location of a specific parcel via the Internet.

Finance Systems Among the oldest information systems are *finance systems*. These systems support the flow of money within and between organizations. Finance systems typically provide accounting functionality to maintain a consistent and auditable set of books for reporting and management support. Another important application of finance systems is the stock market. At a stock market, dedicated information systems are essential to process the operations. Again, the functionality of finance systems is absorbed by ERP systems. The origin of the SAP system, for example, was in finance rather than production planning.

Product Design Systems Enterprise information systems not only support the production of products, they also support the design of products. Examples are *computer-aided design* (CAD) *systems* and *product data management* (PDM) *systems*. CAD systems support the graphical representation and the design of product specifications. PDM systems support the design process in a broader sense by managing designs and their documentation. Typically, there are many versions of the same design, and designs of different components need to be integrated. To support such complex concurrent engineering processes, PDM systems offer versioning functionality.

Workflow Management Systems Many organizations aim to automate their business processes. To this end, they have to specify in which order the activities of a business process must be executed and which person has to execute an activity at which time. A *workflow* refers to the automation of a business process, in whole or in part. Each activity of the workflow is implemented as software. The workflow logic specifies the order of the activities. A *workflow management system* (WfMS) is an information system that defines, manages, and executes workflows. The execution order of the workflow's activities is driven by a computer representation of the workflow logic. The ultimate goal of workflow management is to make sure that the proper activities are executed by the right people at the right time (Aalst and Hee 2004).

Not every business process corresponds to a single workflow. Workflows are case-based; that is, every piece of work is executed for a specific case. One can think of a case as a workflow instance, such as a mortgage, an insurance claim, a tax declaration, a purchase order, or a request for information. Each case is handled individually according to the workflow definition (often referred to as the workflow schema). Examples of business processes that do not correspond to a single workflow are stock-keeping processes; for example, in *make-to-stock* and *assemble-to-order* processes, end products or materials already exist before the order is placed (i.e., before the case is created, manufacturing or assembly activities have already occurred). For this reason, only fragments of such business processes (i.e., in-between stocking points) are considered to be workflows.

Interestingly, WfMSs are embedded in some of the enterprise information systems already mentioned; for example, most ERP and PDM systems include one or more WfMS components. Besides enterprise information systems, middleware software (e.g., IBM's WebSphere) and development platforms (e.g., the .NET framework) embed workflow functionality; see the WebSphere Process Server and the Windows Workflow Foundation. Examples of stand-alone WfMSs are BPM|one, FileNet, and YAWL.

Data Warehouses A *data warehouse* is a large database that stores historical and up-to-date information from a variety of sources. It is optimized for fast query answering. To allow this, there are three continuous processes: The first process extracts data at regular intervals from its information sources, loads the data into auxiliary tables, and then cleans and transforms the loaded data to make it suitable for the data warehouse schema. Processing queries from users and from data analysis applications is the task of the second process. The third process archives the information that is no longer needed by means of tertiary storage technology.

Nowadays, most organizations employ information systems for financial accounting, purchasing, sales and inventory management, production planning, and management control. To efficiently use the vast amount of information that these operational systems have been collecting over the years for planning and decision-making purposes, the information from all relevant sources must be merged and consolidated in a data warehouse.

Whereas an operational database is accessed by *online transaction processing* (OLTP) applications that update its content, a data warehouse is accessed by ad hoc user queries and by special data analysis programs, referred to as *online analytical processing* (OLAP) applications. In a banking environment, for example, there may be an OLTP application for controlling the bank's automated teller machines (ATMs). This application performs frequent updates to tables storing current account information in a detailed format. There may also be an OLAP application for analyzing the behavior of bank customers. A typical query that could be answered by such a system would be to calculate the average amount that customers of a certain age withdraw from their accounts by using ATMs in a certain region. To minimize response times for such complex queries, the bank would maintain a data warehouse into which all relevant information (including historical account data) from other databases is loaded and suitably aggregated.

Queries in data warehouses typically refer to business events, such as sales transactions or online shop visits that are recorded in event history tables (i.e., fact tables) with designated columns for storing the time and the location at which the event occurred. An event record usually has numeric parameters (e.g., an amount, a quantity, or a duration) and additional parameters (e.g., references to the agents and objects involved in the event). Whereas the numeric parameters are the basis for forming statistical queries the time, location, and reference parameters are the *dimensions* of the requested statistics. There are *multidimensional databases* for representing and processing this type of multidimensional data. The leader in the data warehouse market is Oracle (data from 2009).

Business Intelligence Systems A *business intelligence system* provides tools to analyze the performance—that is, the efficiency and the effectiveness—of *running* business processes. These tools extract information on the business processes from the data available in an organization. Different tools and techniques exist, among them business performance management, business activity monitoring, querying and reporting, data mining, and process mining.

Business performance management concentrates on improving the performance of business processes. The goal is to extract information from the history of running business processes and to display this information on a management dashboard. For example, one could monitor a credit approval process to get insight into the length of time required to make the decision.

In contrast to business performance management, *business activity monitoring* aims at providing real-time information on business processes and the activities in these business processes. The goal is to support decision making at runtime. Such a tool may monitor inventory levels, response times, or queues and take action whenever needed.

Querying and reporting tools explore data (e.g., stored in a data warehouse) to provide insight into efficiency and effectiveness of business processes and trends in the environment. Typically, statistical analysis is applied to the data to distinguish between trends and isolated events.

The term *data mining* refers to a collection of techniques to extract patterns from examples. Originally, the term "data mining" had a negative connotation (i.e., data dredging, data snooping, and data fishing), but nowadays data mining is an established research domain with a huge impact. Examples of classical data mining tasks are *classification* (which arranges the data into predefined groups), *clustering* (like classification, but the groups are not predefined), *regression* (which attempts to find a function that models the data with the least error), and *association rule learning* (which searches for relationships between variables). Data mining techniques can be applied to any type of data and do not explicitly consider business processes.

Process mining looks at data from the viewpoint of a particular business process. Information systems usually log the occurrences of events—for example, accepting an order, sending an invoice, or receiving a payment. The availability of such *event logs*, which contain footprints of a business process, enables the discovery of models describing reality. The resulting business process model can be compared with the specification of the business process and used for simulation and performance analysis. Process mining is discussed in section 8.5.

Business intelligence is still a young discipline that will receive more acceptance and attention soon. Most commercial tools support business performance management, business activity monitoring, and querying and reporting rather than the more sophisticated techniques of data and process mining. Business intelligence is so far restricted to reporting information on running business processes and offers little support in terms of how a business process can be improved. The market leader in business intelligence is SAP (data from 2008) with SAP BusinessObjects; other main vendors are SAS, IBM, Oracle, and Microsoft. Examples of open-source projects providing data and process mining software are WEKA (Witten and Frank 2005) and ProM (Aalst, Reijers, et al. 2007).

1.2.3 Enterprise Information Systems in Different Industries

The various types of enterprise information systems have different levels of granularity. For example, SAP Business Workflow is just one component in the large SAP ERP system, but its functionality is comparable to many stand-alone WfMSs. Functionality of software systems is more and more wrapped into services that can be accessed over the Internet, which allows software systems to be viewed at different levels of granularity. Organizations do not develop their enterprise information systems from scratch; they instead purchase large software suites that must to be customized, or they assemble a software system from components. Configuration corresponds to specifying information about the organization and its business processes and to switching functionality on or off. Organizations typically use only a small percentage of the functionality provided by software vendors, such as SAP. Similarly, few hospitals use all of the functionality provided by software vendors, such as ChipSoft and Siemens.

The abundance of functionality in today's enterprise information systems can be explained by looking at the cost of software. Development of enterprise information systems is extremely expensive, because these systems are—from an engineering point of view—highly complicated. However, once developed, software can be copied without much effort. This development cycle is completely different from that of physical products. For this reason, software vendors are tempted to provide an abundance of functionality that can be adapted to the customer's specific requirements. As a result, software vendors shift efforts from software implementation to configuration of enterprise information systems.

The application of a particular enterprise information system and its configuration depends on the industry an organization is operating in. For example, a hospital, a bank, a manufacturer, and a municipality may all use an ERP system, such as SAP, but the configurations will vary. Although all four organizations may use the financial component or the procurement component of SAP, it is likely that only the manufacturer is using the MRP component for production planning. In addition to standard components, these organizations will use industry-specific enterprise information systems. For example, the hospital may use a dedicated radiology information system and an information system to create and maintain electronic patient records. The bank will have software to make calculations related to interest and mortgages, and the municipality will have software to access governmental administrations.

The hospital, the bank, the manufacturer, and the municipality in this example may use the same WfMS (e.g., BPM|one or YAWL), but the workflow schemas that are used to configure the systems of the four organizations are different. For example, the municipality will need to specify the business process for registering a newborn. This business process is irrelevant for the other three organizations.

Given the various types of enterprise information systems and the many ways they can be configured, this chapter does not target specific industries or specific types of enterprise information systems. Instead, we concentrate on general principles of (enterprise) information systems.

1.3 The Life Cycle of an Information System

There are various ways to develop an enterprise information system. Accordingly, the most important question a designer of such a system has to deal with is: how do I develop an enterprise information system? To answer this question, we introduce a *life cycle model* of enterprise information systems. This life cycle model covers the phases of the development process of an enterprise information system. Enterprise information systems are complex software systems that are modified to reflect organizational needs and changes rather than developed from scratch. For this reason, the life cycle model includes phases that address change and redesign of existing enterprise information systems. In this section, we aim at being more generic and consider information systems rather than enterprise information systems.

1.3.1 Introduction to the Life Cycle Model

According to the integrated view of an information system shown in figure 1.1, an information system may include each of the six entities. In definition 1.5, we restricted information systems to software systems, thereby requiring the presence of the technology of figure 1.1. When considering the development process of an information system, however, we interpret the information system in a more narrow sense in which just the software is taken into account. Information systems typically have two development processes. In the first development process, a generic information system is implemented; in the second development process, this system is customized. For example, for an ERP system, software vendors, such as SAP, implement new releases of their ERP system for other organizations. The implementation of the ERP system is guided by the development process of the software vendor. After an organization purchases an ERP system, this ERP system passes through the development process of the organization. In this second development process, the ERP system needs to be installed, configured, customized, and introduced in the organization.

There can be mixtures of these two development processes. For example, the information system of a bank may be composed of selected components of an ERP system and of self-developed software components that provide specific functionality. In this case, the development process for building the information system for the bank includes a software development process similar to that of software vendors. Because of their tremendous complexity, existing information systems are usually redesigned and iteratively improved rather than replaced by a new system. As a consequence, the development process of an information system contains phases, such as maintenance and improvement. For example, in the information system of a bank, the ERP system may be reconfigured or upgraded to a newer version.

Organizations develop and run information systems, which may involve software components purchased from other organizations. People who are going to use the information system are the *users* or *participants*. People who design the information system or the products that are used to assemble the information system are the *designers*. In this section, we concentrate on the work of designers.

Many life cycle models are described in the literature and used in practice. Some aim at the software development process (e.g., within companies), and others aim at the development of an information system in an organization (e.g., a bank). Our life cycle model, depicted in figure 1.2, is a mixture of both. Each rectangle illustrates a phase in the life cycle, and arcs represent the order of the phases. The main cycle models the development process of a new information system. It takes into account the development process of generic software, the development process of information systems that are customized from generic software, and a mixture of these development processes. The two smaller cycles, which contain shaded rectangles, model the development process of existing information systems—that is, the maintenance and the improvement

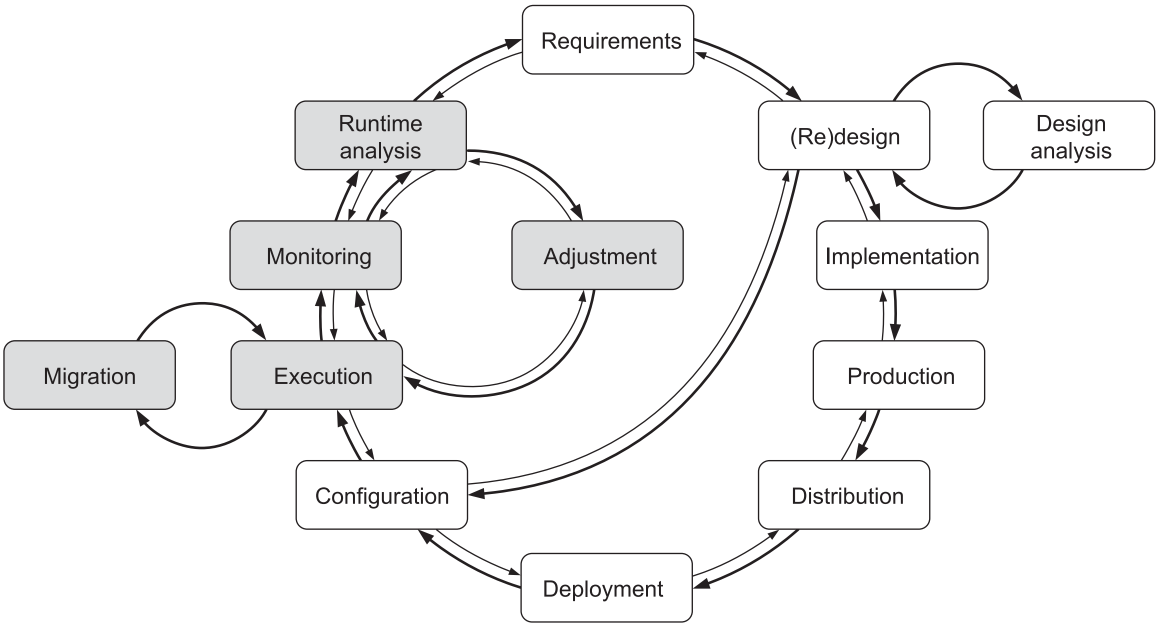

Figure 1.2
The life cycle model of an enterprise information system.

of running information systems. In the following sections, we discuss the life cycle model of figure 1.2 in more detail.

1.3.2 A Software Development–Oriented Life Cycle

The life cycle model in figure 1.2 is based on the observation that information systems are complex, customized (i.e., made-to-order) software systems whose development requires many man-years. Developing an information system can be compared to constructing a tunnel or manufacturing a car. It is usually organized in the form of a project. The main cycle in figure 1.2 specifies the development process of a new customized information system, which is the focus of this section.

We distinguish the following eleven phases for customized information systems: *requirements phase, design phase, design analysis phase, implementation phase, production phase, distribution phase, deployment phase, configuration phase, execution phase, monitoring phase,* and *runtime analysis phase*. Not all of these phases are relevant for all information systems; for example, production, distribution, and deployment phases are only relevant in the case of generic (i.e., made-to-stock) information systems, such as ERP systems, Microsoft Office tools, or database management systems.

Models play an important role in the development process of an information system. A model describes the information system to be designed in a certain form (e.g., textual or graphical). Models can be displayed in many ways, but they are always intended to describe the information system or the business processes supported by it. The way in which such a description is expressed depends on the point of view from which we want to look at the information system and is determined by the purpose of the description. A model abstracts away from aspects that are considered not relevant for the model. There are countless modeling formalisms. Most of them are grounded in logic, set theory, algebra, or graph theory.

Example 1.7 A bicycle map is a model of a geographic area and is intended to support cyclists. Not all aspects of the real landscape are present in the model. The bicycle map represents only those aspects that are important to the cyclists, that is, an overview of all bicycling tracks in the designated area. The map may display the bicycling tracks as blue, even though they have a different color in reality. Only the purpose of the model is important: cyclists want to see bicycling tracks on the map. A map of the same area designed for another means of transport (e.g., car or boat) would look different.

Models may serve as an *abstract description* or as a *specification*. An abstract description model describes an already existing information system. This model allows us to analyze the information system. In contrast, a specification model serves as a specification of what an information system is supposed to do. Such a model is intended to be used for constructing a new information system. The modeling of existing information systems and of information systems to be developed are considered in this book. We investigate business process–related aspects of information systems and use Petri nets extended with data, time, and hierarchy as a modeling formalism.

In the following sections, we discuss the eleven phases of the main life cycle in figure 1.2. The requirements phase and the design phase are of particular interest, because developing models is an essential part of these two phases. We further concentrate on the software development process of information systems. The configuration-oriented life cycle (e.g., configuring a customized information system) is discussed in section 1.3.3.

Requirements Phase The *requirements phase* is the first phase in the main life cycle in figure 1.2. It involves collecting the various requirements for the information system and assembling a coherent requirements specification. In many cases, there is an existing information system that does not satisfy all requirements. It is then wise to analyze thoroughly what the existing information system does for its environment. The result of this analysis will inform which functionality of the existing information system should be preserved in the new one. We can also obtain valuable insights by analyzing the deficiencies of the existing information system and the reasons why a new information system is being developed. After this analysis, we can formulate the requirements of the new information system.

Example 1.8 The requirements phase in the development of a new (simplified) ATM leads to the following requirements. The ATM should allow its clients to query their current account balances and to withdraw money. If clients want to withdraw money, then the ATM should offer them several amounts, but it should also allow them to choose an amount of money. There are several restrictions. For example, the amount of money clients withdraw should be less than a maximum amount (e.g., 500 euros for each day), and it should not lower the client's account balance below a predefined lower bound. Furthermore, if clients just query their current account balance, then their account balance should not change.

The requirements refer to the functionality of the new information system and also to other (nonfunctional) aspects, such as costs, maintenance, and reliability. In the early requirements phase, requirements are expressed in ordinary language. This is important, because key users should be able to understand the requirements. Users typically express requirements in cooperation with designers. In the late requirements phase, requirements are expressed by specification languages and by models resulting in a *domain model* that captures the concepts of the domain under study. The development of requirements specifications is known as *requirements engineering* (Hull, Jackson, and Dick 2004).

Exercise 1.1 Express the main requirements for an information system that advises travelers about a travel scheme (route and time) when they want to make a holiday trip.

Design Phase The purpose of the *design phase* is to develop two models that are suitable to communicate with the users and the software developers of the information system. First, designers derive a *functional design model* from the domain model. The functional design model is expressed in terms of general software modeling constructs, but it still abstracts away from specific implementation. Second, designers derive an *implementation model* from the functional design model by taking the target programming language or implementation framework into account.

A functional design model captures the functionality of the information system. This model typically consists of several diagrams that visualize (static) data models and (dynamic) business process models. It abstracts away from implementation details. This is especially important for the communication between users and designers. Users are laymen and should not be confronted with all details of the information system; instead, end users must understand relevant parts of the model to investigate whether the designer has correctly taken their requirements into account.

Designers can construct a functional design model for an information system in the form of an *executable specification* providing a formal description and a *prototype* of the information system. A prototype is an experimental first version that is used for testing a design and for gaining more insight into the requirements of the information system to be built. It does not normally implement the entire functionality of the information system. For instance, it may lack an ergonomic user interface, it may not provide the necessary security mechanisms, or it may not provide the required performance. At the beginning of the design process, the requirements of an information system are often incompletely and ambiguously specified. By constructing a model and doing experiments with a prototype, ambiguities and hidden requirements may be discovered. This helps ensure that the final information system satisfies the requirements of its users and avoids costly and time-consuming revisions at a later stage.

The second model, which designers construct during the design phase, is an *implementation model*. It is a detailed work design for the software developers who are going to implement the information system. There are usually several work designs, each reflecting a certain aspect or detail of the information system. Because it is essential that the implementation model conforms to the functional design model, the designer has to verify that these models match.

Example 1.9 In the example of the ATM, the designer constructs a functional design model of the ATM on the basis of the previously developed domain model. The functional design model can be an algebraic specification of the static information (e.g., querying the current account balance returns an account balance in euros) and a business process model describing the order of activities (e.g., clients choose to withdraw money, next they can choose between a standard amount or a customized amount, and so on). In addition, the functional design model can contain a prototype showing the user the possible interactions with the ATM. With this model, the user and designer can discuss all open issues of the final design of the ATM.

In the next step, the designer develops an implementation model based on the functional design model. This model may contain detailed information about how the database of the bank must be queried, how the chosen security mechanisms must be implemented, and how the interplay of the information system with the hardware of the ATM must be implemented. The implementation model serves as a basis for discussion between the designer and the software developer to identify the way in which the ATM should be implemented.

In the ATM example, the mediator role of the designer and the benefit of the two models becomes clear. The designer uses the functional design model to communicate with the user and the implementation model to communicate with the software developers.

In the software development process, usually one person or a group of people plays the role of the designer and of the software developer. The distinction between the functional design model and the implementation model may then become blurred. In these circumstances, often the user cannot understand the model because it is too detailed, or the model does not sufficiently support the implementation because it is unclear or incomplete.

Design Analysis Phase The role of the functional design model and the implementation model is not only to serve as a basis for discussion between the designer and the user and between the designer and the software developer. Models abstract away from facts that are considered not relevant for the model, are less complex than the information system, and can, therefore, be analyzed. Analyzing the functional design model and the implementation model is the subject of the third phase of the life cycle, the *design analysis phase*.

The goal of this phase is to gain insight into the model and, hence, into the information system to be implemented. If the model is an abstract description to be used to analyze an existing information system, the model must first be validated. *Validation* checks whether the model correctly reflects the information system. A validated abstract description model and a specification model can be analyzed. There are several ways to analyze a model. *Verification* is an analysis technique to *prove* that the model conforms to its specification. A specification can be another more abstract model or a set of properties that the model must satisfy. Most verification techniques must explore (parts of) the states of the model and analyze whether the desired properties hold in every state. As the functional design model and the implementation model of an information system typically have many states, verification is often hard to achieve. For this reason, another analysis technique is used more frequently: *simulation*. The idea of simulation is to make the model executable and to run certain *experiments* (known as runs or scenarios). A model may allow infinitely many scenarios. Because only a finite number of scenarios can be executed, simulation does not typically visit all states of a model. Consequently, unlike verification, simulation can be applied to verify only the presence of errors but not their absence. Simulation is often applied for *performance analysis*. Performance analysis assesses key performance indicators, such as response time and flow time, to detect possible bottlenecks in the system during the design.

Example 1.10 Using the ATM example, we can specify a scenario of a client who first queries an account balance and afterward withdraws 100 euros. By using simulation, we can execute this scenario on the model and check whether this model behaves as expected. Simulation also allows performance analysis; for example, we could check whether the database system can retrieve the current account balance within a certain time interval. It would be important to verify that clients cannot crash the ATM.

An overview of existing analysis techniques is provided in chapter 8.

Implementation Phase The fourth phase in the life cycle model is the *implementation phase*. In this phase, the information system is constructed. Because an information system is a software system, construction means either programming the entire functionality from scratch or extending or reimplementing existing functionality. Nowadays, software projects increasingly develop generated code. Development tools, such as Eclipse, may generate template code to create a graphical user interface, for instance. The programmer can later modify and refine this generated code. This significantly increases a programmer's productivity.

Production Phase The fifth phase is the *production phase*, in which software of an information system in prepared for distribution. Unlike classical manufacturing processes, it is relatively easy to produce software, because this boils down to copying and downloading. For widely used standard products, such as database management systems and the Microsoft Office tools, however, the production of manuals, CDs, and so forth may be nontrivial. For product software, licensing issues may also require effort.

Distribution Phase In the case of mass production, there is a sixth phase, the *distribution phase*. The goal of this phase is to make the information system available to its future users. The marketing for the information system is also a part of this phase. The production and distribution phases do not apply to customized information systems.

Deployment Phase In the *deployment phase*, the information system is installed in its target environment, and the users of the information system are trained to use it or to work with it. For example, in the case of a health care system in a hospital, professionals must be trained. Training is important in other domains as well, because information systems, such as ERP systems and database management systems, provide a multitude of functionality. The deployment phase is the seventh phase in the life cycle model.

Configuration Phase Many organizations do not implement their information systems from scratch but instead buy standard software, which is often referred to as *commercial off-the-shelf* software or *product* software. In this case, the information system needs to be configured and customized to the organization and its business processes. Even when organizations develop their own software, there is often the need for configuration. This is the subject of the eighth phase in the life cycle model, the *configuration phase*.

For sectors such as financial accounting, inventory management, or production planning, there are customizable standard software packages: ERP systems. These packages have many adjustable parameters, among them the standard currency and the date display format to be used (e.g., 1-Jan-2001, 01/01/2001, or 20010101). The customization of an ERP system can be viewed as a kind of programming in a special language. The difference from conventional programming is that the entire functionality does not need to be programmed. Much of the standard functionality provided can be used instead. Nevertheless, adapting standard business processes to specific organizations may require substantial effort and should not be underestimated. It may even be the case that the standard functionality provided is inadequate, and parts need to be reimplemented.

Execution Phase After the deployment and configuration, organizations can finally run their information systems. In an ideal world, this *execution phase*—the ninth phase in the life cycle model—would be the final phase of the development process, with maintenance consisting of the organization keeping the data up to date and making backups. Because of its complexity, however, it is unlikely that an information system meets all requirements and performs in a way it is expected at the start of this phase. Moreover, the environment of the information system is changing over time. To simplify error detection and to get insight into what functionality is actually used (and also how it is used), information systems log an enormous number of events. These event logs provide detailed information about the activities that have been executed. Event logs play an important role in the successive phases of the life cycle model.

Monitoring Phase In the tenth phase, the *monitoring phase*, organizations extract real-time information about how their information systems perform. Monitoring provides information on the current state of each business process instance and on the performance of the previously executed activities. In a way, the monitoring phase is a simulation of the running business processes in practice. The extracted information can be compared with the domain model (i.e., the requirements) and the functional design model. Unlike the process shown in figure 1.2, the execution phase and the monitoring phase typically run in parallel. The monitoring phase is using data from the execution phase, but it can also influence execution through the adjustment phase (see section 1.3.4).

Runtime Analysis Phase Monitoring is performed while the information system is running, but it is not intended to change information systems or redesign business processes. Monitoring provides relatively simple types of diagnostic information. More advanced analysis techniques are possible and are performed in the *runtime analysis phase*.

In contrast to the design analysis phase in which information system models are analyzed, the runtime analysis phase analyzes whether the implemented information system conforms to its specification. Event logs play an important role in this phase.

These event logs can be analyzed—for example, to figure out whether requirements of the information system are violated—or replayed on the functional design model and the implementation model. Process mining techniques allow information to be extracted from the event logs to provide designers with more insights into the running information system and the supported business processes. Because of the complexity of contemporary information systems and rapidly changing circumstances (e.g., new laws and changing regulations), the importance of the runtime analysis phase is increasing.

1.3.3 A Software Configuration–Oriented Life Cycle

The focus of the previous section was on the development process of information systems that either are generic software systems or contain at least some self-developed software components. The life cycle of these information systems includes, among other phases, an implementation phase and a deployment phase. The production phase and the distribution phase are, in contrast, relevant only for the development process of generic software systems. There are many organizations that do not implement an information system; instead, they construct it from predefined generic software systems, such as ERP systems. In this section, we discuss the life cycle of *customized information systems*, which consists of seven phases of the main life cycle of figure 1.2: *requirements phase, design phase, design analysis phase, configuration phase, execution phase, monitoring phase,* and *runtime analysis phase*.

As for generic information systems, the development process of a customized information system starts with the requirements phase. The designer expresses the identified requirements as a domain model. In the subsequent design phase, the designer derives a functional design model from the domain model and constructs an implementation model. The models are then analyzed in the design analysis phase. In contrast to the development process of a generic information system, the designer uses the implementation model to identify which software components are necessary to create the information system. The purchase of these software components, including actions such as tender procedures, is a process that is orthogonal to the phases of the life cycle. In the life cycle model, we therefore assume that an organization has purchased all necessary software components to design an information system. In the following configuration phase, the designer (supported by software developers) configures and customizes these software components. Recall that configuration refers to choosing between existing predefined parameters and reimplementing some of the standard functionality to adapt it to the requirements of the organization.

The subsequent execution, monitor, and runtime analysis phases are as described section 1.3.2. The monitor and runtime analysis phases are intended to verify that the configuration yields an information system that conforms to the requirements and, hence, to the specification, rather than to verify whether the implemented software components are correct (although these components may contain bugs, like any software). Although generic information systems offer functionality that organizations of different industries can use, configuring an information system such that it perfectly satisfies the requirements of an organization can be time-consuming.

In figure 1.2, it is assumed that the smaller software configuration–oriented life cycle does not include a deployment phase. Deployment is typically not needed if existing information systems are reconfigured; however, if an organization introduces a new enterprise information system, then it must perform the activities mentioned in the deployment phase.

1.3.4 A Runtime-Oriented Life Cycle

Ever-changing market conditions, regulations, and further customizations require organizations to be flexible and to adapt to changing circumstances. As a result, business processes are subject to change. This requires that we adapt the information systems that support these business processes to the new requirements. For this reason, the life cycle model in figure 1.2 includes phases that need to be passed through to change and redesign existing information systems. Shaded rectangles in figure 1.2 resulting in two smaller cycles depict these phases. The first cycle consists of four phases: *execution phase, monitor phase, runtime analysis phase,* and *adjustment phase*. It models that new requirements result in *adjusting* the information system. The second cycle takes this idea of adjusting the information system a step further and addresses the *replacement* of a (part) of the information system with a newer one. This cycle consists of an *execution phase* and a *migration phase*.

Adjustment Phase Monitoring and analyzing a running information system is a continuous process. In the *adjustment phase*, a running information system is adapted to changing circumstances. For example, there may be a new law that clients of a bank are not allowed to withdraw money more than three times a day. Other causes for adjusting an information system are detected errors or performance bottlenecks. In a customized information system, adaptation may change some adjustable parameters. The adjustment phase uses predefined runtime configuration possibilities; that is, the information system is reconfigured but not changed. The cycle in figure 1.2 illustrates that adjusting an information system is also a continuous process. The information system is changed and then again monitored and analyzed.

Migration Phase It is not always possible to adapt a running information system by a reconfiguration at runtime. Changes in the environment may require the replacement of (a part of) an old information system with a new information system. The new information system is developed according to the main life cycle in figure 1.2, as described in sections 1.3.2 and 1.3.3. At a certain point in time, the replacement has to take place. One of the challenges is the migration of data from the old information system to the new one. An example is the business process of a life insurance company.

A new legal regulation may cause a business process to change, while instances of this business process have been running for decades. In this case, each running instance of the old business process has to be migrated to the new business process. This is the subject of the *migration phase*.

1.3.5 Reflection

The development of an information system is, in practice, not as straightforward and well defined as the life cycle model in figure 1.2 may suggest. Typically, we have to revisit previous phases or start over with analysis; that is, the development process is *iterative*. In figure 1.2, this is illustrated by the counterclockwise arcs. In each iteration, the current models are being further refined and extended. As a consequence, the development process is also *incremental*. Phases of the life cycle can overlap with one another. We can construct an implementation model for parts of the information system, even if other parts have already been implemented. This book focuses on the analysis and design phases.

1.4 System Models

In the previous section, we presented a life cycle model of information systems. We explained that, in particular, in the early phases of this life cycle—in the requirements, design, and design analysis phases—models play an important role for specifying existing information systems and for implementing new information systems. In this section, we show that an information system can be viewed as a discrete dynamic system whose behavior can be modeled as a transition system.

1.4.1 Discrete Dynamic Systems

To clarify the most important system concepts, we look at several systems: a laptop computer, a washing machine, a railroad network, a car engine, a turning wheel, a wheel of fortune, a digital alarm clock, and the membership administration of a tennis club. All of these systems possess a state that is subject to change. We refer to them as *dynamic systems*. Dynamic systems can have *discrete* or *continuous* state changes, as described in the following examples.

A laptop computer can switch between four modes: on, hibernate, sleep, and off. A washing machine is washing at one moment and rinsing the next. At the railroad network, a signal is red at one moment, and, a little later, it is green. These three dynamic systems seem to change states in *discrete* jumps.

A car engine consumes fuel continuously and not in discrete quantities; the engine is a *continuous* dynamic system. The rate at which the state changes (e.g., the fuel level) depends on the way the engine is used. A turning wheel of a car is another example of a continuous dynamic system: the wheel turns continuously and not in discrete jumps.

It is a matter of conceptualization whether we consider a system to be continuous or discrete. We can view continuous systems in a discrete manner and vice versa. For example, the state changes of a wheel of fortune seem to be continuous, but only a limited number of states of the wheel matter; namely, those in which the wheel can stop. We can therefore treat the wheel of fortune as a dynamic system with a finite number of discrete states. Similarly, a washing machine can be described as a continuous dynamic system in which the water level is gradually increasing.

We tend to consider the passage of time as a continuous process. Nevertheless, a digital alarm clock is a system that changes states in discrete jumps. The alarm clock halts for a minute at 8:20 a.m. and then jumps to 8:21 a.m. For the function of an alarm clock, it is sufficient to display the time in hours and minutes. In the context of an alarm clock, we may treat the passage of time as a discrete process. This is also the case in other situations—for example, when measuring time at a sports event. For a sports event, we may measure time with higher accuracy (e.g., in milliseconds).

Systems that change states in discrete jumps are *discrete systems*. In mathematics, "discrete" means "distinct" and "noncontinuous." Likewise, systems that change states in continuous jumps are *continuous systems*. Examples of continuous systems are a river, a turning wheel, a chemical reaction, and a flying bullet. A continuous system is typically described using differential calculus. In this book, we do not consider continuous systems; instead, we conceptualize continuous systems as discrete (as with the wheel of fortune example). In the remainder of this book, we refer to a discrete dynamic system as a *system*.

Example 1.11 The membership administration of a tennis club is a system. At any moment, 120 members may be registered. When a new member enters the club, there will be a change in the membership state: the number of members increases to 121. This is a discrete change, even if the secretary of the club does not immediately import the data of the new member.

Exercise 1.2 Consider a bank where we are only interested in the balances of all bank accounts. Explain why this is a system.

There are two important concepts for describing a system: *state* and *state change*. Between two successive instantaneous changes, nothing happens in a system. We say that the system is in a certain state. A state change is a *state transition* or *transition*.

Example 1.12 A fan is a system with two states (off and on) and two transitions. One transition changes from off to on and the other from on to off.

1.4.2 State and State Space

We now take a closer look at the state concept of a system using an example. The medicine cabinet of a hospital department contains three kinds of drugs: painkillers, sleeping pills, and antipyretics. Nurses supply these drugs to patients as needed, so the stock decreases each time medication is administered. From time to time, the stock is replenished. Each moment, the medicine cabinet is in a certain state; that is, it contains a certain amount of each of the three drugs. We can record the actual stock in a table, such as table 1.1.

Table 1.1 gives a description of the state of the medicine cabinet at a particular moment. Two things are important for such a state description. First, we only include information in a state description that is relevant for the system. Second, there can be several descriptions of a state.

There is information that is not represented in the state description of the medicine cabinet—for example, the current temperature in the cabinet, the size of the cabinet, and whether the painkillers are on the first or the second shelf. What is relevant depends on the purpose of the system and on the needs of the users of the system. For the nurses, it is not interesting to know where a drug is exactly stored in the medicine cabinet. Because only three kinds of drugs are stored, it is possible to quickly see where each drug is. For a hospital pharmacy, which stores thousands of drugs, on the other hand, drug location would be important.

Depending on the needs of the users, there are several ways to describe the possible states of a system. Table 1.2 shows an alternative description of the contents of the medicine cabinet. For each drug, we define a desirable minimum number—that is, its *base stock*—and represent the difference of the actual stock from this number. The sum of these numbers equals the actual stock.

A state description should represent all things that may change and whose change is relevant for the system. The state description of the medicine cabinet, for instance, provides the number of drugs in stock. This is the only relevant information for this system. Other aspects of the medicine cabinet, such as the history of a drug stock, are not relevant and are abstracted away.

Table 1.1
The Contents of the Medicine Cabinet

Type of drug	Actual stock
Painkiller	14
Sleeping pills	9
Antipyretics	8

Table 1.2
An Alternative Representation of the State of the Medicine Cabinet

Type of drug	Base stock	Difference
Painkiller	10	4
Sleeping pills	10	−1
Antipyretics	5	3

Exercise 1.3 A hobbyist made a wheel of fortune from the wheel of a bicycle with 36 spokes. A simple but smart mechanism makes sure that the wheel can be stopped between each two spokes that are next to each other. Describe all possible states of this wheel.

In the following discussion, we assume that we always deal with state descriptions that represent states in an adequate way (corresponding to the interests of the users of the system and at the right abstraction level). We use the term "state" without explicitly mentioning that a certain description is involved.

A system can be in several states. The set of all possible states is the *state space*. The number of possible states of a system can be large. If the maximum stock of the medicine cabinet were 19 painkillers, 19 sleeping pills, and 19 antipyretics, there would be $20 \cdot 20 \cdot 20 = 8,000$ possible states.

To specify a state space, we use the notation of mathematical set theory. The fan in example 1.12 can be in states off and on. Accordingly, we formally represent the state space S of the fan as the set:

$S = \{\mathsf{off}, \mathsf{on}\}$.

For the possible states of the medicine cabinet, only the actual stock of the drugs is relevant. We can represent the actual stock as a triple (i.e., a sequence of three elements). The state displayed in table 1.1 is then represented as $(14, 9, 8)$. The state space is too large to be easily enumerated, but we can define it as:

$S = \{(x, y, z) \mid x, y, z \in \{0, \ldots, 19\}\}$.

Recall that, in a set expression, the order of elements does not matter. From the previous expression of state space S, we can conclude nothing about the order in which the states occur.

Exercise 1.4 Describe the state space of the wheel of fortune in exercise 1.3. How can you formally represent the state space?

1.4.3 Transitions and Transition Systems

A system can stay in the same state for a short or a long time, but it normally changes from one state to another after a certain time. The state change is performed instantaneously. For example, when a nurse takes drugs from the medicine cabinet, the stock decreases. The state of the medicine cabinet changes through such an *atomic* (i.e., indivisible) action.

For the time being, we abstract away from the time that is needed for a transition. This is not a problem for industrial applications. In the case of the medicine cabinet, we are interested in the changing stock of drugs and not in the time necessary to take out drugs from the cabinet. In the case of the fan (see example 1.12), we are interested in whether the fan is on or off and not in the relatively short time it takes to change from one state to another. If state changes take considerable time (i.e., they are nonatomic), then we can split the state change into two state changes: one indicating the start of the state change and the other indicating the completion of the state change. For example, we can split a transition "repair_car," indicating a car repair, into transitions "start_repair_car" and "end_repair_car," indicating the start and the completion of the car repair.

During a *transition*, a system changes from one state to another. We are not interested in what exactly happens during this change. For this reason, we can write a transition as an ordered pair:

(old_state, new_state).

Assume that the fan is switched on. The ordered state pair

(off, on)

describes exactly what is going on. First, we mention the old state and then the new state. The pair (on, off) represents the transition when the fan is switched off.

Definition 1.13 (Transition) A *transition* is an ordered pair (x, y) in which x and y are elements of the state space S—that is, $x, y \in S$.

Exercise 1.5 At a certain moment, the medicine cabinet contains three painkillers, five sleeping pills, and eight antipyretics. Then a nurse takes two sleeping pills and three antipyretics from the cabinet. Express this transition as an ordered pair of states.

For every system, we can record each transition with an ordered pair of states. If we consider all possible transitions of a system, then we obtain a set of ordered pairs of states.

Example 1.14 Consider the model of the ATM that we described in example 1.8. The state space is represented as:

$S = \{$idle, card, pin, balance, money,
offer, choice, payout, violation, output_card$\}$.

The following set of transitions is possible:

$TR = \{$(idle, card), (card, pin), (pin, balance), (pin, money), (money, offer),
(money, choice), (offer, payout), (offer, violation), (choice, payout),
(choice, violation), (violation, money), (violation, output_card),
(balance, output_card), (payout, output_card), (output_card, idle)$\}$.

The ATM is initially in state idle. A client then inserts a bank card (yielding state card) and keys a pin (pin). Next, a client can either query an account balance yielding state balance or withdraw money yielding state money. In state money, a client can either choose an amount of money (choice) or select an offered amount of money (offer). If the chosen amount of money is not too high, the money is paid out (payout), and the ATM returns the card to the client (output_card). Otherwise, the ATM enters state violation, from which the menu can be reached (state money), or the client asks the ATM to return the card (output_card). From state output_card, the ATM moves to state idle from which it can serve the next client.

In mathematics, a set of ordered pairs is a (binary) *relation*. Accordingly, each system has a *transition relation*, which contains all possible transitions of the system. The identifier *TR* denotes the transition relation. A transition relation usually does not contain all ordered pairs of states that can be formed by combining two states, because some pairs are not possible. In the case of the ATM, for example, state pairs (card, idle) and (money, balance) do not represent possible transitions.

We obtain the set of all ordered pairs of states of a state space S by forming the Cartesian product $S \times S$. For a state space $S = \{a, b, c\}$, the Cartesian product consists of $3 \cdot 3 = 9$ elements. We write:

$$S \times S = \{ (a,a), (a,b), (a,c), (b,a), (b,b), (b,c), (c,a), (c,b), (c,c) \}.$$

For the ATM, the Cartesian product of the state space with itself has $10 \cdot 10 = 100$ elements, but not all of these elements correspond to a possible transition. The transition relation *TR* is consequently a subset of the Cartesian product:

$$TR \subseteq S \times S.$$

By specifying the state space S, the transition relation *TR*, and an initial state, we can describe a system. The *initial state* of a system is the state in which a system starts its operations.

Definition 1.15 (Transition system) A *transition system* is a triple (S, TR, s_0) where S is a finite state space, $TR \subseteq S \times S$ is a transition relation containing all possible state changes, and $s_0 \in S$ is the initial state.

The notion of a transition system in definition 1.15 is similar to the definition of a finite automaton (Hopcroft and Ullman 1979). A finite automaton is a transition system in which every transition is labeled by a symbol from a given alphabet. For simplicity, we do not label transitions, but it is easy to add a labeling function to (S, TR, s_0) assigning a label to all elements in TR.

In definition 1.15, we restricted ourselves to transition systems with a finite set of states. If this set does not contain too many states, state space and transition relation can be depicted in a diagram. We draw for each possible state a rectangle with rounded corners and for each transition an arrow from the old state to the new state. Figure 1.3 shows the transition system of the ATM. Such a diagram is the *state-transition diagram* of a transition system. An incoming transition without source pointing to state idle denotes that the ATM is initially in state idle (i.e., $s_0 =$ idle). The notion of a state-transition diagram is similar to the notion of a state diagram in other notations, such as the Unified Modeling Language (UML) (Rumbaugh, Jacobson, and Booch 1998; Object Management Group 2005).

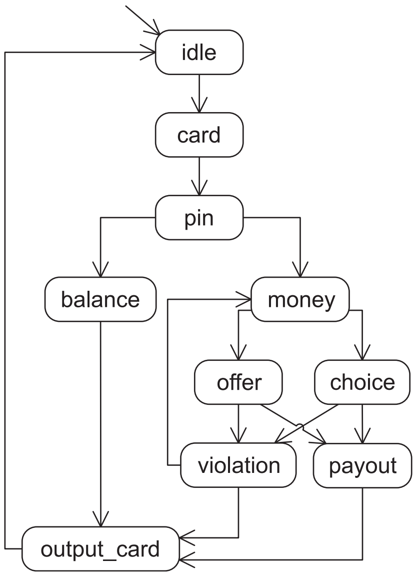

Figure 1.3
A state-transition diagram of the ATM.

The general concept for structures, such as state-transition diagrams, is a directed *graph*. A directed graph consists of nodes that are connected by directed edges.

Definition 1.16 (State-transition diagram) A *state-transition diagram* is a directed graph in which the nodes represent the states of the transition system, and the directed edges represent the possible transitions.

Exercise 1.6 A simple elevator system serving a building with five floors can be considered to be a discrete dynamic system. Reason why the elevator can be seen as a discrete dynamic system, define its transition system, and draw the state-transition diagram. Assume that the elevator is initially at the ground level.

In the example of the medicine cabinet, for which we must deal with 8,000 possible states, it is not feasible to depict the transition system as a state-transition diagram. The diagram would be too large and unmanageable. There are techniques to visualize large transition systems, but these techniques provide only an impression of the topology of the state space.

1.4.4 Transition Sequences and the Behavior of a System

Thus far, we have considered single transitions in isolation. To study the behavior of a system, we must consider possible sequences of transitions and the states visited by these sequences. It is important to determine which states can be reached from a given initial state of the system.

Definition 1.17 (Reachable state) A *reachable state* is a state that the system can reach from the initial state after zero or more transitions.

Question 1.18 The initial state of the ATM in example 1.14 is state idle. Are all states reachable?

We can determine reachability by following the arrows in the state-transition diagram in figure 1.3. State idle can be reached after zero transitions. From state idle we can go to state card, next to state pin, to state balance, to state output_card, and to state idle again.

A sequence of states reached by following the arrows in figure 1.3 is a *transition sequence*. For example,

⟨idle, card, pin, money, choice, payout, output_card, idle⟩

is a transition sequence that represents withdrawing money by selecting a specific amount of money. The transition sequence for querying the account balance is:

⟨idle, card, pin, balance, output_card, idle⟩.

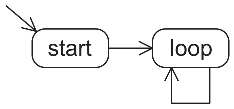

Figure 1.4
A self-loop: State loop can change to itself.

Transition sequences give us insight into the dynamics of a system. On the basis of the transition sequences, we can see how the system behaves. The set of all possible transition sequences from a given initial state specifies the *behavior* of the system.

In the behavior of a transition system, two special cases are possible. The first case is a state in which no further transition is possible. Such a state is a *terminal state*, often referred to as deadlock. The state-transition diagram of the ATM in figure 1.3 does not have a terminal state, but if we remove transition (output_card, idle), then state output_card becomes a terminal state. A terminal state should not be confused with a state that can only change to itself, known as a *self-loop*, as shown in figure 1.4. In the state-transition diagram in figure 1.3, there is no self-loop.

The second special case is the situation in which, from every state, at most one transition is possible. This means that, in the state-transition diagram, each node has at most one outgoing arrow. Such a system is *deterministic*, whereas a system with multiple options for a successor state is *nondeterministic*. The transition system of the fan (see example 1.12) is an example of a deterministic system. The ATM is a nondeterministic system, because from state pin a client can choose state balance or state money.

Exercise 1.7 Describe why the medicine cabinet is a nondeterministic system.

1.5 Roles of Models

A model of an enterprise information system can serve different purposes. We identify two dimensions to characterize models of enterprise information systems. First, a model may be *design oriented* or *analysis oriented*. Second, a model may be *information system oriented* or *business process oriented*. Table 1.3 shows that these two dimensions are orthogonal, and therefore we have to distinguish four kinds of models.

The role of a *design/information-system-oriented model* is to create a new enterprise information system or to adapt an existing one. An example is a specification model and an implementation model. The purpose of such a model is to specify desired functionality, to document requirements and design decisions, to guide implementation efforts, or to prescribe a configuration.

An *analysis/information-system-oriented model* focuses on the analysis of an enterprise information system. The purpose of such a model is to gain insight into an existing

Table 1.3
The Different Roles That Models Play

	Design	Analysis
Information system	Specification	Verification
	Requirements	Performance analysis
	Design decisions	
	Implementation	
	Configuration	
Business process	Business process reengineering	Performance analysis
	Continuous process improvement	Gaming

enterprise information system or into an enterprise information system after it has been created. For example, we can verify models to discover design flaws in new or existing enterprise information systems, and simulation can predict performance under different circumstances.

In contrast to the two previous kinds of models, a *design/business-process-oriented model* focuses on the (re)design of business processes supported by an enterprise information system (and not on the enterprise information system). The goal is to improve the performance of business processes by redesigning them. Business process reengineering (BPR) (Hammer and Champy 1993) aims at a radical redesign of business processes. New enterprise information systems may be needed to achieve such a redesign. Consequently, BPR initiatives may trigger enterprise information system development. Continuous process improvement (CPI) (Harrington 1991) aims at less radical change. Business processes are continuously reviewed to search for gradual improvements. Continuous monitoring can, for example, support CPI initiatives.

An *analysis/business-process-oriented model* focuses on the analysis of business processes supported by an enterprise information system. The goal is to evaluate existing business processes or to judge alternative business process designs obtained through BPR/CPI initiatives. Simulation is the primary analysis technique used here. It can evaluate the performance of business processes (e.g., response times, flow times, inventory levels, costs, and service levels). Furthermore, interactive simulation models can create management games that help users to spot inefficiencies and understand new ways of working.

In some cases, it may not be clear whether a model is intended to be used for information system analysis and design or for business process analysis and design. Later in this book, we shall see examples of models that are used to analyze the performance of business processes using simulation but that also serve as a specification or configuration model for an information system. To simulate a system, additional information about times and probabilities is necessary. For example, we need to know the time a system stays in each of its states. Without time information, we cannot analyze the performance of the system. If the system is nondeterministic, we need information about the *probability* that the system chooses a particular successor state. Such information can be easily added to a system model or can be extracted from historical data.

Example 1.19 The state-transition diagram in figure 1.3 models the functional design of an ATM. To simulate this model, we need to add time information and probabilities. As an example, reading the bank card in state idle may take two seconds, checking the pin code in state card five seconds, and returning the bank card to the client in state payout three seconds. In state pin, the client may choose in 70% of the cases withdrawal of money and in 30% an account balance. After adding probabilities to all transitions, we can simulate this model and, for example, calculate the probability that the ATM enters state violation.

Markov models (Marsan et al. 1995; Haas 2002; Haverkort 1998) extend transition systems with time and probabilities. Like transition systems, Markov models suffer from the *state explosion problem* (Valmari 1998); that is, even small industrial systems have far more states than a computer can handle and therefore cannot be verified. For this reason, we must use higher-level models and simulation techniques (Buzacott 1996; Marsan et al. 1995).

Exercise 1.8 A wristwatch is an example of a simple information system. Give an estimation of the number of possible states and transitions of a watch that contains only information about the present time. A possible state of this watch is, for example, 23:11:55.

It is unrealistic to model enterprise information systems in terms of a transition system (S, TR, s_0). The state space is too large to be enumerated or to be captured in a simple mathematical set expression. Finding a suitable representation for the transition relation is even more difficult. Accordingly, we need more powerful notations.

The state space S is typically captured using complex data types or database tables. To model the structure of the state space, *data models* are used. Examples are class diagrams in UML (Rumbaugh, Jacobson, and Booch 1998; Object Management Group 2005), entity-relationship (ER) models, crow's foot diagrams, and natural language information analysis method/object role modeling (NIAM/ORM). Data modeling is concerned with the identification of the relevant types of data entities and their relationships. Beside *data modeling*, terms such as *information modeling* and *object modeling* are in use.

Data models do not capture behavior. We therefore need *process models* to describe the transition relation TR. Examples of process modeling techniques are Petri nets, UML activity diagrams, BPMN, and EPCs. In this book, we concentrate on process modeling and use Petri nets extended with data, time, and hierarchy as a formalism.

A *system model* refers to the state space and to the transition relation. Consequently, data and process modeling are required to create such models.

1.6 Test Yourself

1.6.1 Solutions to Exercises

1.1 The information system should allow the specification of a journey from one location (home) to a holiday destination. It should be possible to make the trip by bicycle, boat, car, train, airplane, or bus (or a combination of these modes of transport). The traveler should be able to indicate when the journey should start and end. Given the preferences of the traveler, the information system should propose several alternatives for the travel scheme, such as the cheapest, the fastest, the shortest, and the easiest possibilities.

1.2 This system is dynamic, because it is subject to state changes in the form of changing balances through money transfers. It is discrete, because the changes happen instantaneously; that is, a money transfer is an instantaneous event.

1.3 The spokes of the wheel could be numbered 1 to 36 in such a way that there is always one number between two spokes. When the wheel stops between two spokes, its state can be identified by its number.

1.4 The state space consists of 36 states in which the wheel can stop. These states are numbered 1 to 36. In addition, there is one more state the wheel can be in: the wheel could be turning and not in one of the states 1 to 36. Formally, we denote this as:

$S = \{\text{turning}, 1, 2, 3, 4, \ldots, 36\}$.

We could also represent state *turning* as a number—for example, as 0.

1.5 We can describe the old and the new state by the triple $(3, 5, 8)$ and $(3, 3, 5)$, respectively. The pair $((3, 5, 8), (3, 3, 5))$ represents the transition.

1.6 The elevator system is dynamic, because it does not stay in one state but jumps from one floor to the next. The system is discrete, because the elevator goes step-by-step from one floor to the next. The state space of the elevator system is a set with five states, one for each floor:

$S = \{0, 1, 2, 3, 4\}$.

If we assume that the elevator may go up or down one floor at a time, then the transition relation TR is as follows:

$TR = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 3), (3, 2), (2, 1), (1, 0)\}$.

Assuming that the initial state is $s_0 = 0$, figure 1.5 depicts the state-transition diagram.

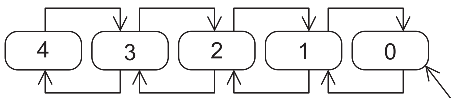

Figure 1.5
State-transition diagram for the elevator system.

1.7 The medicine cabinet is a nondeterministic system, because it is composed of states from which more than one transition is possible. Drugs can be removed from the cabinet or replenished in various numbers and combinations.

1.8 The state 23:11:55 corresponds to the point of time five seconds before twelve minutes past eleven at night. The number of possible states of the wristwatch is equal to the number of points of time (counted in seconds) in one day. The number of possible states is $24 \cdot 60 \cdot 60 = 86{,}400$. Because there is exactly one transition possible from each state, the number of transitions is also equal to 86,400.

1.6.2 Further Exercises

Exercise 1.9 Explain the terms “business process” and “information system.”

Exercise 1.10 List as many types of enterprise information systems as you can and give one characteristic feature or an example for each of them.

Exercise 1.11 Explain the life cycle of developing a new information system and the life cycle of redesigning an existing information system. Use the life cycle model in figure 1.2.

Exercise 1.12 Draw the state-transition diagram for a washing machine with state space

$S = \{\text{off, defective, pre-wash, main\_wash, rinse, whiz}\}$

with $s_0 = \text{off}$ and transition relation

$$\begin{aligned} TR = \{&(\text{pre-wash, defective}), (\text{main\_wash, defective}), (\text{rinse, defective}),\\ &(\text{whiz, defective}), (\text{off, pre-wash}), (\text{pre-wash, rinse}),\\ &(\text{rinse, main\_wash}), (\text{off, main\_wash}), (\text{main\_wash, rinse}),\\ &(\text{rinse, off}), (\text{rinse, whiz}), (\text{whiz, off})\}. \end{aligned}$$

Exercise 1.13 The behavior of the washing machine in exercise 1.12 is not completely realistic, because transition sequences

$\langle \text{off, pre-wash, rinse, off} \rangle$

and

⟨off, main_wash, rinse, main_wash, rinse, main_wash, rinse, . . .⟩

are possible. Adjust the state space and the transition relation such that these transition sequences are no longer possible. Draw the improved state-transition diagram.

Exercise 1.14 A Dutch traffic light is an example of a system with three possible states: R (red), G (green), and O (orange). Model a T-junction with three traffic lights (see figure 1.6) as a transition system and draw the state-transition diagram. The traffic lights are programmed in such a way that at least two lights are red at the same time—that is, for at most one direction of the traffic, the traffic light can be green or orange. (Hint: Represent each state by a combination of three colors.)

Exercise 1.15 To improve the traffic flow, the traffic light system of exercise 1.14 is upgraded to a situation with five lights; see figure 1.7. The goal is to program the traffic light system such that crossing cars coming from different directions cannot have a green light at the same time. Furthermore, if at any point lights turn orange, they must first turn red before other lights can change. Model this system as a transition system and draw the state-transition diagram.

Exercise 1.16 The score in a game of tennis is calculated as follows. The first player who wins four rallies in total and at least two rallies more than the opponent wins the

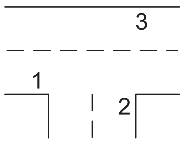

Figure 1.6
A T-junction with three traffic lights.

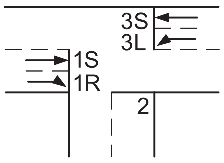

Figure 1.7
A T-junction with five traffic lights.

game. If both players have won three rallies (40–40), then the player who wins the next rally gets the advantage. If this player wins another rally, she wins the game. If she loses the next rally, she loses the advantage, and the score is equal to the situation in which both players have won three rallies (40–40). Model this system as a transition system and draw the state-transition diagram.

Exercise 1.17 Consider the process in a restaurant. After customers enter the restaurant, a waiter assigns them a table and gives them the menu. The customers then order, and the waiter writes down this order, takes the menu, and delivers the order to the kitchen. A cook prepares the order, and the waiter brings it to the customers who then consume it. If the appetite of the customers is not yet satisfied, they can call the waiter and ask for the menu again (after which the entire process repeats itself). If the customers are satisfied, they call the waiter and ask for the check. When that check arrives, they pay and leave.

1. Give the transition system that models the behavior of a customer and the transition system that models the behavior of a waiter. Draw both state-transition diagrams.
2. If you want to make one state-transition diagram in which you describe the behaviors of a customer and a waiter, how many states do you need?

Exercise 1.18 Figure 1.8 depicts a simplified remote control of a TV. It has six buttons to choose a channel, one button to regulate the volume, one button to mute the sound (and to turn it on again), and one button to switch the TV on or off. We consider the remote control and the corresponding TV as a system and assume that the possible states of this system are controlled by the buttons on the remote control.

1. Is this system a discrete dynamic system?
2. Describe all possible states of this system.

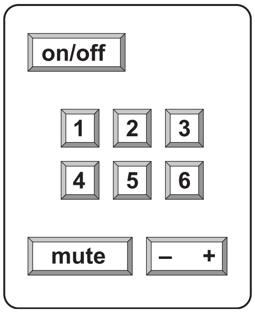

Figure 1.8
The remote control of a TV.

3. The transition relation is too large to be depicted as a state-transition diagram. Give examples of possible and impossible transitions. Pay attention to switching the TV on and using the volume button in combination with the mute button.

Exercise 1.19 Consider a transition system with state space

$\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

and with transition relation

$\{ (0, 1), (1, 2), (2, 3), (3, 4), (3, 5), (5, 0), (5, 4), (4, 4), (6, 7), (7, 6), (7, 8) \}$.

1. Draw the state-transition diagram.
2. Which states are reachable from the initial state 0?
3. List three transition sequences that start in state 0.
4. Does the system have a terminal state?

1.7 Summary

In this chapter, we introduced information systems in general, took a more detailed look at enterprise information systems, and characterized important types of enterprise information systems. We described the different phases in the life cycle of developing and maintaining information systems. We then showed that an information system is a discrete dynamic system whose behavior can be modeled as a transition system. Finally, we discussed the four roles that models play.

We also introduced transition systems as the simplest technique to model discrete dynamic systems and business processes. In the next chapters, we present more advanced modeling techniques that facilitate the modeling of complex enterprise information systems and the business processes they support.

After studying this chapter, you should be able to:

- Explain the terms “information system” and “business process.”
- List the most important types of enterprise information systems and briefly characterize them.
- Describe which life cycle phases are required to develop and to maintain an information system.
- Explain the terms “discrete dynamic system,” “state,” “state space,” “transition,” “state-transition diagram,” “transition sequence,” “deterministic system,” and “nondeterministic system.”
- Describe a system as a transition system and represent it in the form of a state-transition diagram.
- Determine the transition sequences of a simple transition system.
- Explain the difference between a data model and a process model.
- Explain the four roles that a model of an enterprise information system can play.

1.8 Further Reading

There are many books on information systems in the literature. Alter (2002) introduces information systems, their development, and modeling of information systems from a business perspective. Weske (2007), in contrast, concentrates on business processes and business process management. Van Hee (2009) investigates formalization aspects and the integration between data and process modeling.

In this book, we emphasize the role of process models in the realization of enterprise information systems. WfMSs are information systems that are directly driven by business process models. We therefore reference several books on workflow management systems—see work by Van der Aalst and Van Hee (2004), Dumas, Van der Aalst, and Ter Hofstede (2005), Ter Hofstede et al. (2010), Jablonski and Bussler (1996), and Leymann and Roller (1999)—and elaborate more on this in the next chapter.

There are many life cycle models for devloping an information system. Two classical life cycle models are the *waterfall model* and the *spiral model*. The waterfall model, presented by Royce (1970), was the first publicly documented life cycle model. The model was developed to cope with the increasing complexity of aerospace products. The model's focal point is on documentation. The spiral model was introduced by Boehm (1988) to address problems associated with the waterfall model. Alter (2002) introduced a life cycle model that concentrates more on the management perspective and less on the software.

The final topics addressed in this chapter are discrete dynamic systems and transition systems. Hopcroft and Ullman (1979) provide a good overview of this topic. For a deeper study of data modeling, we refer to ER modeling of Chen (1976) and UML (Rumbaugh, Jacobson, and Booch 1998; Object Management Group 2005).