2

An Introduction to Conceptual Modeling of Information Systems

Antoni Olivé

2.1 The Functions of an Information System

The concept *information system* began to emerge around 1960. Even though it may be considered an old concept, it is still difficult to define what an IS is. Part of that difficulty is because ISs can be analyzed in at least three distinct and complementary perspectives [1]:

- The contribution they provide;
- Their structure and behavior;
- The functions they perform.

From the first perspective, ISs are defined as means for wider systems to achieve their objectives. That kind of definition emphasizes that ISs are subsystems of wider systems, to which they contribute. An IS does not exist for itself. Examples of that kind of definition would be: "An IS is a system designed to support operations, management, and decision-making in an organization" or "An IS is a system that facilitates communication among its users."

For our purposes, the main problem with that kind of definition is that it does not give a clear characterization of ISs. The wider system of which an IS is part may require means that are not ISs to achieve its objectives. Furthermore, other things can provide the same type of contribution, without being an IS. For example, there are various ways to facilitate communication among users, including working physically close to each other or participating in meetings.

Even if it is difficult to define ISs in terms of the contribution they provide, it is important to realize that this perspective is essential during their development. The requirements of an IS are determined from the objectives of the organization for which the system is designed and built.

From the second perspective, definitions emphasize the structure and the behavior of the physical and abstract elements that make up an IS. Both structure and behavior can be characterized at different levels of detail.

For the purposes of conceptual modeling, the most useful definitions are those based on the functions performed by ISs, that is, definitions that emphasize what ISs do, abstracting from why and how they do it.

Within this third perspective, the classical definition says that "an IS is a system that collects, stores, processes, and distributes information." That definition is commonly accepted for both its simplicity and its generality. However, some comments may be in order to make it more precise.

First, in IS engineering, we should restrict the definition to designed systems, that is, systems an engineer designs and builds [2]. The restriction is needed because natural systems that perform information-processing functions are beyond the scope of our study. For example, in cognitive science the human mind is viewed as a complex system that receives, stores, processes, and distributes information.

Second, the definition is too general with respect to the kind of information an IS may deal with. In fact, the definition poses no constraint on the kind of information, with the result that it encompasses systems that many people would not consider ISs. For example, a fax could be considered an IS according to that definition, because it can be seen as a system that receives documents (which contain data representing some information), stores them (even if only for a short time), translates them (i.e., changes the representation of the information), and sends the result through the phone.

The usual constraint on the kind of information handled by an IS is that it must be about the state of some domain (also called object system or universe of discourse). The nature of this domain is irrelevant to the definition of IS. For many systems, its domain is an organization, but the

definition does not exclude other, different domains, such as a vehicle, the atmosphere, or a chess game. According to that definition, a fax machine is not an IS. A fax does not consider the documents it sends as information about the state of some domain. To a fax, documents are just uninterpreted data.

Thus, we define an IS as a designed system that collects, stores, processes, and distributes information about the state of a domain. It is easy to agree on those functions, but the problem is that they are too general and are not related to the purpose for which the IS exists. For those reasons, many authors prefer a more specific definition of the functions, one that captures more neatly the nature of ISs.

To that end, it is considered that an IS has three main functions (Figure 2.1) [3]:

1. *Memory function,* to maintain a representation of the state of a domain;
2. *Informative function,* to provide information about the state of a domain;
3. *Active function,* to perform actions that change the state of a domain.

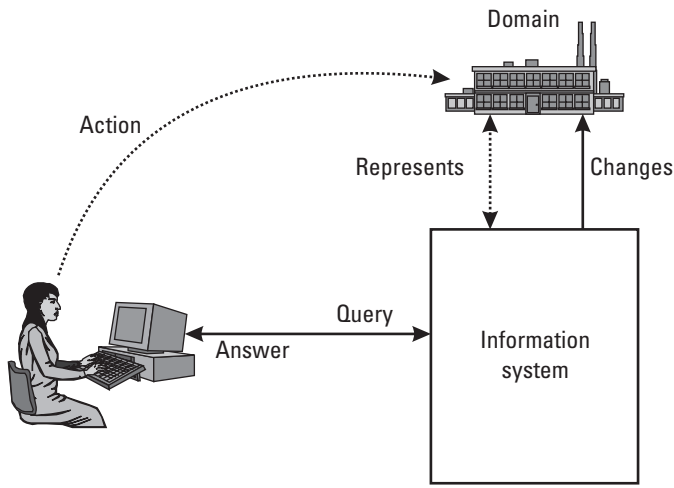

Figure 2.1 The functions of an IS.

2.1.1 The Memory Function

The memory function is needed by the other two functions. Its objective is to maintain internally a representation of the state of the domain. Moreover, the representation must be correct and complete [4].

The memory function can be performed in two execution modes: *on request* or *autonomously*. In the first mode, the system memorizes the state because a user explicitly tells the system the state and orders (normally implicitly) the system to memorize it. For example, a system knows customers' addresses because any time a customer changes addresses, a user tells the IS the new address and expects the system to remember it. The IS has no way to know customers' addresses except by a user explicitly telling it that information.

In the second mode, *autonomously*, the system memorizes the state of the domain without an explicit request from a user. This mode has two variants. In the first variant, the system is able to observe autonomously the state, for example, a system that periodically reads a device that gives the temperature of a building. In that case, the system can maintain a representation of the temperature because it gets it directly from the environment. The second variant is related to the active function and will be described later.

The memory function is considered to be passive, in the sense that it does not perform anything that directly affects users or the domain. However, it is required by the other functions and constrains what they can perform.

2.1.2 The Informative Function

With the informative function, the system provides users with information about the state of the domain. Often, the state of the domain is observable directly in the domain, and at the same time it is represented in the IS. For example, the quantity of a given product in the shelves of a retail store may be observed when necessary, and at the same time it can be represented in the IS. In those cases there is a redundancy, but it is a desired one, because it may be easier to ask the system than to observe the domain.

In other cases, the state is represented only in the IS, and it is not possible (or difficult) to observe it directly in the domain. For example, in a retail store it is not possible to observe how many units of some product have been sold up to a given moment. As another example, in the banking domain, consider balances of accounts. The balance of an account at a given instant cannot be known by observation of the owner of the account or the bank office where the account was created. The only place where balances are

represented is inside the IS. In those cases, the IS is the only source of information about the state, and the system becomes indispensable to its users.

To perform the informative function, the system needs an inference capability, allowing it to infer the required information from the memory. In the most frequent case, users pose a query, and the system answers it. Both query and answer are in a language understood by the users and by the system.

Queries may be extensional (the most frequent case) or intensional. An extensional query asks information about the state of the domain (either the current state or some previous one), and the system gives an extensional or intensional answer. An extensional answer, which is the usual case, consists of more or less elaborated information about the state of the domain. Examples of simple extensional answers are:

- Joan takes the Algebra course.
- Eighty students take the Programming course.

Some extensional answers must be much more elaborated and may require a statistical analysis, a simulation or the execution of a decisional model. Examples of such answers are:

- Ninety percent of customers that buy books also buy CDs.
- No customer has bought more than 200 books.

Less frequently, the answer to an extensional query may be intensional. An intensional answer characterizes the state of the domain, but it does not describe the state explicitly [5]. For example, to the question "Who earns more than $100,000?" the system might answer, "The managers."

Intensional queries ask about the kinds of information the system knows, rather than particular information [6], for example,

- What do you know about students?
- What is the maximum number of courses a student may take simultaneously?
- What is a student?

The informative function also admits the two execution modes. The most frequent case is the *on request* mode: Users get information when they ask it explicitly. In contrast, in the mode *autonomously*, users define a condition on the state of the domain and order the system to inform them when that condition is satisfied. For example, the condition might be "the temperature is over 40°C" and users expect that the system will issue some signal when that condition is satisfied.

The informative function does not change the state of the domain. The system merely provides the information requested by users. It is the users who will take actions that change the domain, if they want to do so.

2.1.3 The Active Function

With the active function, the system performs actions that modify the state of the domain. To perform the active function, the system must know the actions it can take, when those actions can be taken, and their effect on the state of the domain.

The active function also has the two execution modes. In the *on request* mode, users delegate to the system the taking of some action that may modify the state of the domain. For example, users may ask the system to calculate the interests to be paid to bank accounts and charge them to the accounts' balances.

In the mode *autonomously*, users delegate to the system the taking of some action that may modify the state of the domain, when some condition is satisfied. The system will monitor the state of the domain, and when the condition is fulfilled, it will perform the delegated action.

The nature of actions that may be delegated to the system (in both execution modes) are varied. It may be a simple and well-defined action or one for which only the objectives are defined, leaving autonomy to the system on how to achieve those objectives.

The classical example of active function, with mode *autonomously*, is the automatic replenishment of a store. Users define, for each product, a reorder point and a quantity to order. The system maintains the quantity on hand of each product, and users delegate to it to issue orders to suppliers when the quantity on hand is below the reorder point. It is assumed that orders to suppliers are part of the domain and, thus, the state of the domain changes when a new order is issued.

It is interesting to note that, in the preceding example, if the orders were not part of the domain, then the automatic replenishment would not be an example of the active function. It would be an example of the informative

action. Outputs from the system could be a requirement for action to the users, but the state of the domain would not be altered.

Given that the active function may change the state of the domain without the direct intervention of users, it is a function that calls the memory function, in the execution mode *autonomously*.

Table 2.1 summarizes the examples of the three functions and the two execution modes.

2.1.4 Examples of ISs

All conventional ISs perform a memory function and some informative function. We will not describe any concrete examples, since they are well known, and identification of the functions they perform is simple. However, it may be worthwhile to comment on some particular classes of systems and to see that, even if one might doubt whether they are ISs, they in fact perform the functions we have seen in this section.

Example 2.1

Assume a chess-playing system that can play against either a human or a machine. We are going to see that this system may be considered an IS.

The domain consists of the board, the pieces and their position on the board, the player, and the rules of the game. At any moment, the domain is in some state, which is time varying. The rules of the game, however, are fixed. The system has to maintain a representation of the state of the domain; otherwise, it would not be able to play. When a move is completed, the system must somehow know that in order to update the state representation. This is a simple example of the memory function of an IS.

The system has to visualize on the screen the state of the game continuously. When a game starts, the system shows the initial distribution of pieces.

Table 2.1
Examples of Functions and Execution Modes

Functions	**Execution Modes**	
	On request	Autonomously
Memory	Change of a customer's address	Reading of temperature
Informative	Courses a student takes	Signal when temperature is over 40°C
Active	Pay interests to accounts	Automatic replenishment

After every move, the system must show the new distribution. It is therefore an example of the informative function, in the mode *autonomously*.

Once the player has made a move, it is expected that the system will think about the available alternatives to achieve its objective (using the current state of the game and the knowledge the system may have) and that, after a while, it will make its own move. In making the move, the system changes the state of the domain. Therefore, this is a complex example of the active function.

If the system were naive enough to offer sincere advice on the next move to the player, that would be an example of the informative function, in the mode *on request*.

Example 2.2

Let us consider an e-mail system. The domain consists of users, who can send or receive messages, distribution lists, active messages, folders created by users to organize their active messages, and so on. Each message has a content, a subject, a sender, a date, and one or more receivers. Normally, the content and the subject of a message are uninterpreted data for the system.

The memory function consists of maintaining a representation of the state of the domain. The main part of the state will be represented only within the system, and it is not directly observable in the domain. The state changes when a user issues a message, receives a message, creates or deletes a folder, puts a message in a folder (or removes a message from it), and so on.

Among other things, the informative function allows users to visualize their active messages (at different levels of detail), as well as the contents of their folders.

The active function consists of sending messages issued by users to their receivers. The effect is that the sent message is put in the input folder of each receiver. This function is performed in the mode *on request*.

Example 2.3

This last example is not a concrete system, but a class of systems: real-time systems. There is not a consensus on what real-time systems are, but they tend to be identified by means of a set of common characteristics [7].

First, a real-time system monitors and controls an environment (i.e., it issues controlling commands that change the environment). Using our terminology, monitoring the environment is a memory function, and controlling it is an active function. Second, real-time systems interact with users for whom they perform a needed function. Such functions may be either informative or active. Real-time systems frequently have various sensors and

intersystem interfaces that provide continuous or periodic input. These are the mechanisms by which the system knows the state of the environment, for the memory function. Finally, a real-time system has a set of actuators or intersystem interfaces that must be driven periodically. They correspond to the mechanisms by which the system sends to the environment the output form, the active function.

A real-time system has other characteristics that do not refer to the essential functions that must be performed but to how they must be performed, for example, sampling intervals of sensors, response time, concurrent processing of multiple inputs, high reliability, resource (main or secondary memory, processor capacity, etc.) limitations, and so on. These characteristics are important, but they do not change the fact that real-time systems may be seen as ISs.

2.2 Conceptual Modeling

Section 2.1 reviewed the main functions of an IS. To be able to perform those functions, an IS requires some knowledge about its domain. The main objective of conceptual modeling is the elicitation and formal definition of the general knowledge about a domain that an IS needs to know to perform the required functions.

This section describes the kinds of knowledge required by most ISs. The line of reasoning we will follow is this:

- If the memory function of an IS has to maintain a representation of the state of the domain, then we must define which is the concrete state that must be represented.
- The state of most domains is time varying, which requires defining the causes of changes and the effects of those changes on the state.
- The representation of the state in the IS must be consistent; therefore, it is necessary to define what it means to be a consistent representation.
- Many times, answering queries posed by users requires some inference capability on the part of the IS. This capability uses derivation rules, which must be defined.

This section develops that line of reasoning. Along the way, we introduce informally the terminology and give an intuitive idea of the basic concepts.

2.2.1 Conceptual Schema of the State

The objective of the memory function of an IS is to maintain a consistent representation of the state of its domain. The state of a domain consists of a set of relevant properties.

The question of which exactly are the relevant properties of the domain of an IS depends on the purpose for which the IS is built. We have already mentioned that an IS is always a means for a wider system to achieve its objectives. The relevant properties are determined by the functions of those objectives and of the expected contribution of the IS to them. We focus here on what are relevant properties rather than how to determine them. That, of course, does not mean that the latter aspect is less important than the former one.

In the IS field, we make the fundamental assumption that a domain consists of objects and the relationships between those objects, which are classified into concepts. The state of a particular domain, at a given time, consists, then, of a set of objects, a set of relationships, and a set of concepts into which those objects and relationships are classified. For example, in the domain of a company, we may have the concepts of *Customer*, *Product*, and *Sale*. Those concepts are usually stable. On the other hand, at a given instant, we have objects classified as customers, objects classified as products, and relationships between customers and products classified as sales.

That fundamental assumption is also shared by disciplines such as linguistics, (first-order) logic, and cognitive science. Unfortunately, those disciplines have not yet arrived at an agreement in the terminology, the definitions, the concepts, and the mechanisms to distinguish among objects and relationships in a domain. The result is that we do not have at our disposal a solid theoretical basis, and, as is often the case in the IS field, we must adopt a humble and eclectic attitude.

The assumption that a domain consists of objects, relationships, and concepts is a specific way to view the world (domain). At first sight, it seems an evident assumption. Reality, however, is far from that. Other views are possible, views that may be more adequate in other fields. As a simple and well-known example in propositional logic, one assumes that domains consist of facts, which may be either true or false. The study of the nature and

the organization of the real world is the subject of the branch of philosophy called *ontology*.

When we assume that a domain consists of objects, relationships, and concepts, we commit ourselves to a specific way of observing domains. The term used in ontology to designate such commitments is *ontological commitment*. In the IS field, the term *conceptual model* is the commitment corresponding to viewing domains in a particular way. In principle, the same conceptual model can be applied to many different domains, and several conceptual models could be applied to the same domain.

The set of concepts used in a particular domain is a *conceptualization* of that domain. The specification of that conceptualization, in some language, is called an *ontology* of that domain [8, 9]. There may be several conceptualizations for a given domain and, thus, several ontologies. An ontology is also a concrete view of a particular domain. Therefore, it is also an ontological commitment for the persons that observe and act on that domain. In the IS field, ontologies are called *conceptual schemas*, and the languages in which they are written are called *conceptual modeling languages*.

As we will see, conceptual models of ISs are much more complex than simply assuming that a domain consists of objects and relationships. A conceptual model assumes that a domain includes other "things," and also that objects, relationships, and concepts have several properties that must be distinguished. On the other hand, a conceptual model includes a view of how a domain changes.

There is a great diversity in conceptual models, which make them more or less useful in particular situations or for particular purposes. However, all of them share the fundamental assumption we have mentioned and that we will make precise.

We begin trying to establish the distinction between concept and object. According to the dictionaries, a *concept* is "an abstract or generic idea generalized from particular instances" or "an idea or mental picture of a group or class of objects formed by combining all their aspects."

Those definitions fit our purpose. A concept, then, is something that we have formed in our mind through generalization from some instances. A concept has an extension and an intension. The extension of a concept is the set of its possible instances, while the intension is the property shared by all its instances.

As human beings, we use the concepts we have to structure our perception of a domain. In that sense, concepts are like eyeglasses with which we observe a domain. Concepts allow us to classify the things we perceive as

exemplars of concepts we have. In other words, what we observe depends on the concepts we use in the observation.

Classification is the operation that associates an object with a concept. The inverse operation, *instantiation*, gives an instance of a concept. The set of objects that are an instance of a concept at some time is called the *population* of that concept at that time.

An *entity type* is a concept whose instances are individual and identifiable objects. Objects that are instances of an entity type are called *entities*. Figure 2.2 shows a simple example of entity and entity type.

All entities are instances of some entity type, but an entity may be an instance of more than one entity type. For example, in Figure 2.2 the entity shown could also be an instance of *Doctor*.

If there is a "thing" in which we are interested, but we are not able to classify it in any of the concepts we have, then we have to form a new concept of which that "thing" could be an instance. In contrast, there may be concepts without instances in the usual domains. The typical example is *Unicorn*. In conceptual modeling, we do not show interest in concepts without instances.

Some concepts are associative, in the sense that their instances relate two or more entities. *Relationship types* are concepts whose instances are *relationships*. Figure 2.3 shows an example of relationship type *Reads* between *Person* and *Book*.

A particular case of relationship is the *reference relationship*. In principle, each entity in the domain must have at least one name that allows us to distinguish among entities. A name is a linguistic object that we use to refer to an entity. Names are also entities and, therefore, instances of some entity

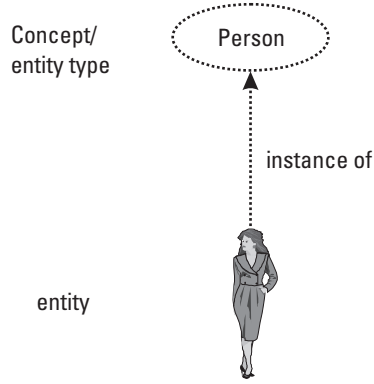

Figure 2.2 Entities as instances of concepts.

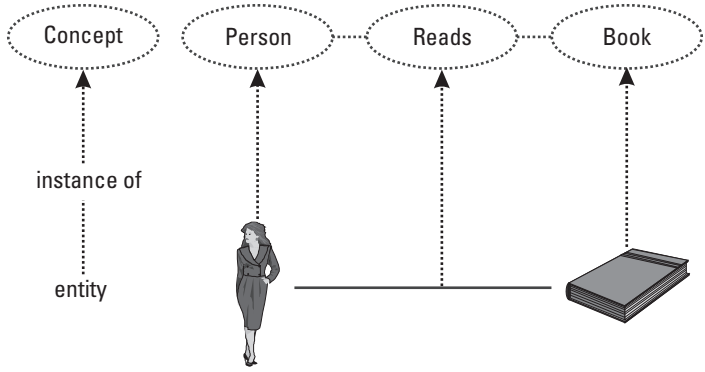

Figure 2.3 Relationships as instances of concepts.

type, that we call lexical entity types. Examples of lexical entity types are *String*, *Number*, *Bar code*, *ISBN code*.

The correspondence between an entity and its name is established by means of a relationship. That relationship is also an instance of some relationship type, sometimes called a reference relationship type. There may be more than one reference relationship type, for example,

- The relationship type between *Person* and *String;*
- The relationship type between *Book* and *ISBN code;*
- The relationship type between *Book* and *Number* (in a library).

The set of entity and relationship types used to observe the state of a domain is the conceptualization of that state. The description, in some language, of that conceptualization, as well as other elements we will see in a moment, is called the ontology of the state, or the *conceptual schema of the state*. The set formed by the conceptual schema of the state and the conceptual schema of the behavior, which will be described later, is called the *conceptual schema*. Languages used to define conceptual schemas are called conceptual modeling languages.

Not all entities and relationships in a domain need to be represented in the IS. That leads us to distinguish between conceptual schema of a domain and conceptual schema of an IS. The former describes the conceptualization of the domain, without regard to which entities and relationships will be represented in the IS. In contrast, the latter describes only the fragment of the

conceptualization such that its entities and relationships are represented in the IS.

2.2.2 Information Base

An *information base* is a description of the entities and relationships of a domain that are represented in the IS [10]. In principle, this description could be done in any language, but usually it is done in a logic-based language. Sometimes, the description of an entity or a relationship is called a fact, and we say that the information base contains the facts about a domain.

For example, if we use the language of first-order logic as the modeling language, we could have a schema formed by predicates *Person*, *Book*, and *Reads*, which represent entity types *Person* and *Book* and relationship type *Reads*. The information base might contain, at some time, the facts *Person(A)*, *Book(B)*, and *Reads(A,B)*.

Figure 2.4 illustrates the relationship between a conceptual schema and an information base. The conceptual schema of the domain includes the concepts from Figure 2.3 and two other concepts, *Town* and *Lives*. However, we want to represent in the IS only the entities and relationships shown in the conceptual schema of Figure 2.3. The conceptual schema is described in a graphical language, in which rectangles correspond to entity types and lines to relationship types. The information base contains three facts, described in the language of first-order logic. Predicates correspond to entity and relationship types.

The information base does not exist physically. It is only an abstract description we use to reason about a schema and to exemplify particular situations in a domain. Naturally, the IS must maintain an internal

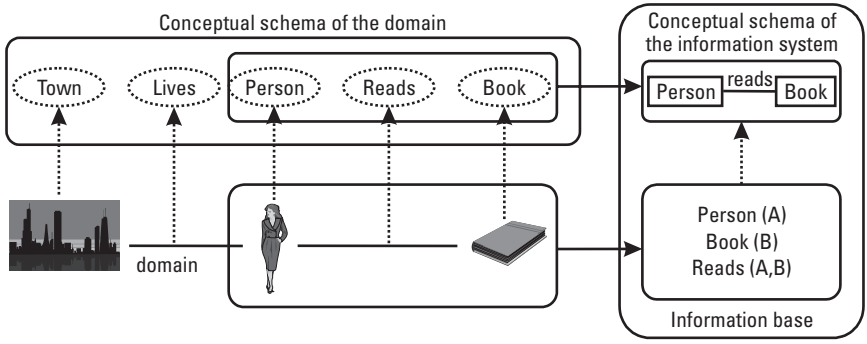

Figure 2.4 Conceptual schema and information base.

description (e.g., in a DB) of the entities and relationships in a domain, but this description is at a very low level in order to be efficiently processable by the processors used. The information base is a description close to the conceptual schema and is not meant to be an internal description.

Unfortunately, the term *conceptual model* is not always used with the same meaning in the literature. Besides the meaning we have given to it, other meanings we may find are these:

- Conceptual model = conceptual schema
- Conceptual model = conceptual schema + information base

We have chosen to use three distinct terms (*conceptual model*, *conceptual schema*, *information base*) to distinguish three different concepts. The same distinction is well established in the DB field, where we distinguish clearly among data model (for instance, relational data model), DB schema (in some data models), and DB (instance of a schema).[1]

2.2.3 Conceptual Schema of the Behavior

Most IS domains change through time, at two levels: conceptual schema and state. Changes at the conceptual schema level are less frequent than those at the state level, and their origin is due to changes in users' interests: For whatever reason, users lose interest in the representation of some entities and relationships or they want other entities and relationships to be represented in the IS.

The most frequent changes (and the only ones we consider here) occur at the state level. It is easily observable that the state of most IS domains changes through time. In consequence, if the information base is a truthful representation of that state, then the facts of the information base will need to change through time.

1. The term *information base* may be confused with the term *knowledge base*, used in the deductive DB's field, as well as in artificial intelligence. A knowledge base is a set of representations of knowledge about a domain [11]. Normally, the language used to represent this knowledge is the language of first-order logic. The knowledge may be simple facts, which are represented as atomic formulas, or general knowledge about a domain, which is represented as complex formulas. In conceptual modeling, the general knowledge about a domain is represented in the conceptual schema, while simple facts are represented in the information base. Therefore, the correspondence is knowledge base = conceptual schema + information base.

We say that there is a change in the state of the domain at time t if the entities or relationships that exist at t are different from those existing at the previous time. In other words, a state change is a change in the population or one or more entity or relationship types between two states: the new state (corresponding to t), and the old state (corresponding to $t-1$).

Any change in the population of an entity (relationship) type can always be decomposed into a set of one or more elementary changes of the following types:

- Insertion of entity (relationship). This change happens when there is an entity (relationship) in the new state that did not exist in the old state.
- Deletion of entity (relationship). This change happens when there was an entity (relationship) in the old state that does not exist in the new state.

The causes of the changes are the *events* [3, 12]. A domain does not change its state if no event happens. An event is any circumstance that happens at a given instant and whose effect is a change in the domain state. Normally, these circumstances are actions (or decisions) performed by human beings that act on a domain (e.g., hiring an employee or making a move in a chess game), but they also may be the result of physical processes (e.g., dropping some amount of liquid into a tank or the rising of the sun).

It is usually assumed that events are instantaneous, that is, they do not have duration. It is also assumed that an event causes a transition in the domain, from an old state to a new one, without any noticeable intermediate state. In many cases, those assumptions do not pose any particular problems. For example, the process of hiring a new employee takes some time, but it is likely that we are interested only in the outcome of that process: From that moment on, the person will be an employee, which he or she was not at the previous time.

In some cases, however, events have duration. To handle those cases in conceptual models that require instantaneous events, it may be necessary to refine the conceptual schema of the domain or the event itself. For example, assume the domain includes the relationship type *Is at* between persons and places. In principle, it seems natural to consider that persons are at some place at any moment. Let us consider now the event corresponding to the move of a person from an origin to a target. If we assume that the event is instantaneous, then the person will continue to be at some place at any

moment. But if we assume that a move is not instantaneous, then there will be a temporal interval during which we will not know where a person is. If we want to take into account that fact, we will need to do the following:

- Refine the conceptual schema of the domain: now there will be some times when we do not know where a person is.
- Transform the move event into two events: the beginning and the end of a move.
- Consider that the effect of the beginning of a move is that we enter a state in which we do not know where the moving person is.
- Consider that the effect of the end of a move is that there is a relationship between the moving person and the target place.

For the designer, it is important to distinguish between external and generated events. An event is external if it occurs independently of the IS. If the IS is computer-based, external events happen even if the system is out of service. Such events are called external because they happen outside the control of the system. The system will need to be notified of the events (to update its information base), but the system itself has not produced the events. Many events are external, for example, the hiring of an employee or the sunrise.

A system may know external events either by direct observation or by users' communication:

- In direct observation, the system has some mechanism that allows it to detect the occurrence of events. For example, a system may have a sensor that detects the arrival of a car in a toll station.
- In users' communication, the users tell the system of the events when they occur. For example, when a company changes the price of a product, the system is also notified of the change.

As mentioned in Section 2.1, an IS may also have an active function. In the active function, the users may delegate to the system the generation of some events that change the state of the domain when some conditions hold. A generated event is an event induced directly by the IS. Without the participation of the system, the event would not be generated, and, therefore, the domain would not change. The system may generate an event as a response to an explicit request from users, when it detects that the state of the domain

satisfies some condition, or because it considers the event necessary to achieve an objective defined by the users.

Example 2.4

Assume an IS that controls an elevator. At each floor there is a button that users can press to request the elevator. Pressing one of the buttons is an external event. The system responds immediately by turning on the light associated with the button (to inform users that the system is aware of their request). Turning on and off light buttons are generated events. Taking into account the current position of the elevator, as well as the pending requests, the system issues several commands to start or stop the motor. Those commands are also generated events.

Events, either external or generated, are also instances of concepts. An *event type* is a concept whose instances are events. Events may have relationships with other entities. In particular, all events have a relationship with an entity that is a time instant, which corresponds to the time when the event happens. Figure 2.5 shows an example of the event type *change of residence*. Events of this type are related with a person (who changes), a town (new residence), and a date (occurrence time).

The set of event types that exist in a domain is part of the conceptual schema of events. The description, in some language, of that schema, as well as other elements described next, is called the *conceptual schema of the behavior*.

To be able to update the information base, the IS must know not only the events that have happened but also their effect on the information base.

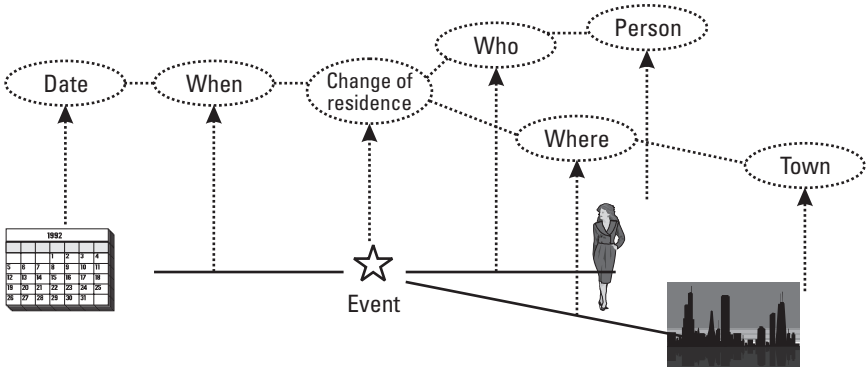

Figure 2.5 Event type and instance.

The definition of that effect, in some language, is also part of the conceptual schema of the behavior.

In conceptual modeling, there are several distinct ways to define the effect of events. The most usual way consists of defining, for each event type, an effect function that gives the new state for any old state and any instance of the event type. For example, the effect function corresponding to the event type *change of residence*, shown in Figure 2.5, might be (informally)

> If an event of type *change of residence*, of person *p* and town *c*, occurs on date *d*, then in the new state, corresponding to date *d*, person *p* will not live any longer where she lived before, and she will be living in town *c*.

In the example, the effect of the event is quite limited, and the effect function is simple. In practice, however, it is not so easy to define the effect, because there are many event types, and some of them have a complex effect function.

For generated events, the conceptual schema of the behavior includes the definition of the generating conditions, that is, when the events must be generated.

Example 2.5

Assume an IS that monitors the level of water in a tank. The system has a sensor that detects the level of water at any time. It is expected that the system will keep the input valve open when the water level is below a desired minimum and closed when the level is above a desired maximum. Generated event types are the opening and the closing of the valve. The generation condition of the former could be "when the current level is below the minimum and the valve is not open already" and that of the latter, "when the current level is above the maximum and the valve is not closed."

2.2.4 Integrity Constraints

The information base is a representation of the state of the domain. An IS obtains and updates the information base from messages received through the input interface or by direct observation of the domain.

In a perfect world, the information base would be an exact representation of the domain. Input messages would always be correct, and the system would receive all relevant messages. Furthermore, the direct observation of the domain would always be faithful. In a perfect world, the representation would always be correct (or valid) and complete.

Unfortunately, in the real world it is likely that some received messages are incorrect, in the sense that they communicate something that is not true. Also, the direct observation of the domain may be distorted. In such cases, some of the facts in the information base may not be valid. It is also likely that the system does not receive all relevant messages; then the information base may not be complete.

Validity and completeness are the two components of the integrity of an information base [13]. We say that an information base has integrity when all its facts are valid and it contains all relevant facts. Integrity is an important property of an information base. Lack of integrity normally has negative consequences, which in some cases may be serious.

In most systems, total integrity can be achieved only by human intervention. In many cases, it is necessary to check the facts in the information base against the domain. For example, many retail stores need to check periodically that the products they have on shelves correspond to their records in the IS. It is not difficult to see that in some cases the cost of integrity is high and hardly avoidable.

However, it is possible to build mechanisms in the IS that automatically guarantee some level of integrity. We can define conditions on the information base such that, if satisfied, we can have some level of confidence on its integrity. These conditions, called *integrity constraints*, are defined in the conceptual schema. An integrity constraint is a condition that might not be satisfied under some circumstances, but it is understood that the IS will include mechanisms to guarantee its satisfaction at any time.

Example 2.6

Assume that a conceptual schema has a relationship type *Assigned to*, involving entity types *Employee* and *Project*. Suppose that in the domain all employees are always assigned to one or more projects. An integrity constraint might be "all employees are assigned to some project." Once defined in the conceptual schema, we can assume that all states of the information base will contain for each known employee at least one relationship with a project. The constraint, however, does not guarantee total integrity (e.g., the information base could have wrong assignments), but its satisfaction is a necessary condition.

We say that an information base is *consistent* if it satisfies all defined integrity constraints. We also say that a constraint is violated when the information base does not satisfy it. When a constraint is violated, the system must produce some response to maintain consistency. The most frequent case is when a violation is caused by the arrival of some erroneous message,

and the response is usually the rejection of the message, asking for its correction.

Most integrity constraints refer to facts of the information base, and then they are part of the conceptual schema of the state. Some constraints, however, refer to events; then they are part of the conceptual schema of the behavior. An example of the latter, which refers to events of type *Birth*, could be "a person cannot be parent of himself."

2.2.5 Derivation Rules

By means of the informative function, an IS provides information about the state of the domain to users, either when they request it or under predefined circumstances.

If an IS does not have any inference capability, it can provide only information collected from the environment. In some cases, that may be all that is required, but in most cases users expect that systems have some capability to infer new facts from the ones they know. A simple example is totaling. If we give to the system a sequence of numbers, we normally assume the system will at least be able to compute their total.

Most ISs have some inference capability, which requires two main components: derivation rules and an inference mechanism. Derivation rules are defined in the conceptual schema. The inference mechanism uses derivation rules to infer new information. How the inference mechanism works may vary from one IS to another, and it is considered to be part of the internal structure of the system; therefore, it is not specified in the conceptual schema.

A *derivation rule* is an expression that defines how new facts may be inferred from others. The concrete form of this expression depends on the conceptual modeling language used. Often, the expressions are formulas in a logic style, but nothing prevents the use of conventional algorithms. For example, assume we want to define the derivation rule corresponding to the concept *grandparent* from the concept *parent*. An expression in logic style would be "a person *gp* is grandparent of person *gc* if *gp* is a parent of a person *p* and *p* is a parent of *gc*."

An equivalent algorithmic expression that gets the four grandparents of person *gc* could be:

1. Get the two parents *p1* and *p2* of *gc*.
2. Get the two parents *gp1* and *gp2* of *p1*.

3. Get the two parents *gp3* and *gp4* of *p2*.
4. The grandparents of *gc* are *gp1*, *gp2*, *gp3*, and *gp4*.

Derivation rules may be specific of a given domain (e.g., a bank), applicable to all domains of a certain class (e.g., banking), or domain independent (e.g., statistical concepts). The conceptual schema must include all derivation rules that can be used in a particular system, but we should explicitly define only those rules that are specific to our domain. The other derivation rules could be shared by all conceptual schemas for domains of the same class or by all conceptual schemas.

In practice, most derivation rules infer new facts of the information base, and then the rules are included as part of the conceptual schema of the state. However, nothing prevents the inference of events from other events, and then the corresponding derivation rules are part of the conceptual schema of the behavior. For example, a derivation rule referring to events of type *Travel* could define *Long travels* as those travels such that the distance traveled is greater than 1000 km.

2.3 Abstract Architecture of an IS

Section 2.2 presented conceptual schemas. This section shows the essential role these schemas play in the architecture of ISs. By architecture, we mean the main components and their relationships. In principle, there are many possible architectures, and choosing the most convenient for a particular IS depends on many factors, including the preferred architectural style and the hardware and software platform on top of which it must work. However, we do not need to take such diversity into account here. For our purposes, it will suffice to consider the ANSI/SPARC abstract architecture proposed in the ISO report [10] (Figure 2.6).

To illustrate this architecture and the role played by conceptual schemas in it, we will use the example of a chess-playing system that can play with persons or with other systems.

The conventional representation of the state of a chess game is a drawing like the one shown in Figure 2.7. However, not everybody uses exactly the same representation; different icons can be used to denote the same piece. Some users may prefer other graphical representations (e.g., the three-dimensional view), and in some cases text-based representations may be preferred (e.g., in machine-machine communication).

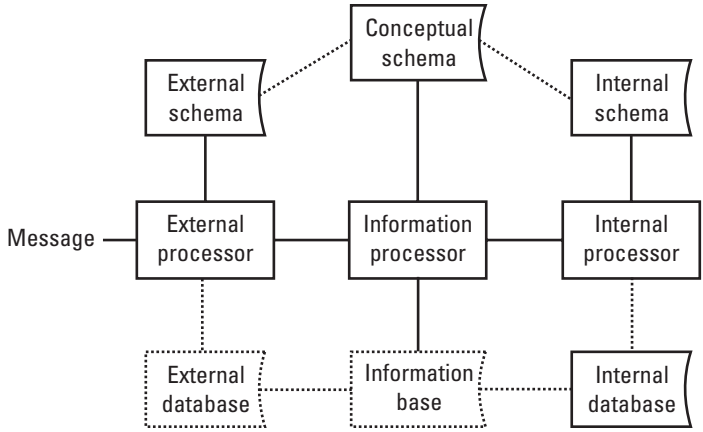

Figure 2.6 ANSI/SPARC architecture.

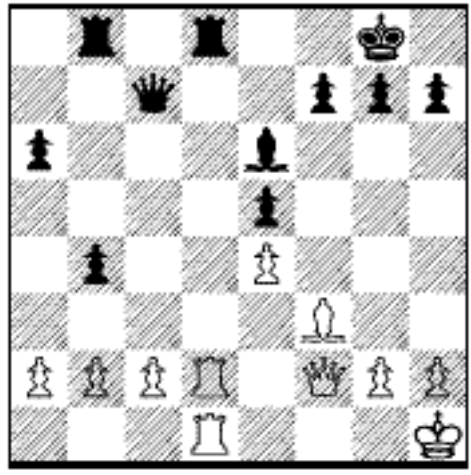

Figure 2.7 A representation of the state of a chess game.

An external schema is a form of representation of the state of the domain used in the domain, and an external DB is the representation of the state of the domain in that external schema. Figure 2.7 can be considered an external DB. External DBs are virtual, in the sense that they do not have a physical and persistent existence within the system.

Besides a form of representation, external schemas also include aspects of manipulation of this form, like the language used to ask queries or to

communicate external events. In the example, we again find some diversity. There are several textual (official) and graphical ways to represent a move (e.g., as a string, like the string "D71," or by dragging a piece to the desired place).

The result is that in general there are several external schemas for a given domain and it is not possible to single out one that satisfies all possible users and all possible uses. Therefore, the system must deal with several external schemas. To do that, the system needs to know the meaning of the representations used and the meaning of the allowed manipulations.

Figure 2.8 shows a simplified conceptual schema of the example. In the figure, entity types are represented by rectangles and relationship types by lines connecting the involved entity types. The name of the relationship type is placed near the line, with a small filled triangle that shows the way to read the name.

Each piece is of some type (king, queen, bishop, etc.), has a color (black or white), and is located at some square. Squares also have a color. For clarity, we will call board square (or just square) to a square that is part of the board, and representation square to a square drawn in the representation of the board (external schema). A board square is located at a row and a column,

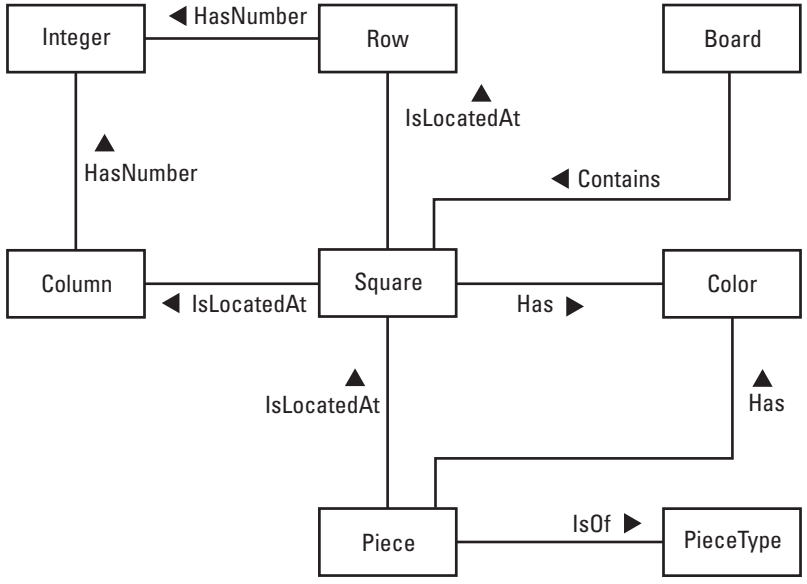

Figure 2.8 Conceptual schema of the chess-playing example.

which define its position in the board. Rows and columns have a number (integer).

The conceptual schema might also include a derivation rule defining that a board square is *free* if there is not a piece located at it; otherwise, it is *occupied*.

There is a single conceptual schema and there may be one or more external schemas. External schemas are defined in terms of the conceptual schema. For instance, the correspondence between the conceptual schema in Figure 2.8 and the external schema used in Figure 2.7 is as follows:

- The board is represented by a (large) square, subdivided into 64 smaller representation squares corresponding to the board squares.
- Representation squares are painted with the same color as the corresponding board squares.
- Each piece has a different icon, depending on its type and color.
- If a piece *p* is located at a board square *s*, then the icon corresponding to *p* is put over the representation square corresponding to *s*.

The correspondence between manipulations and the external events is defined similarly. For example, when the user drags a piece to a representation square, the conceptual meaning is a move of that piece to the board square where it is released.

The *external processor* is the architectural component that interacts with users. In principle, there is an external processor for each external schema. The external processors receive the messages from users (in the language of the external schema), translate them into the language of the conceptual schema, and forward them to the information processor.

The *information processor* is the component that handles the (conceptual) messages originated by the users and performs the active function that may be delegated to the system. In particular, if a message communicates an external event, then the information processor has to apply the corresponding effect function and check that the resulting state is consistent. In the case of the example, if a new move is received, the information processor has to check whether the move is valid and, if so, to update the state of the game.

To perform those tasks, the information processor needs to access and manipulate the state of the domain. It cannot use an external representation because, in general, they may be partial, and, on the other hand, they include aspects that do not have any relationship with the nature of the domain.

For example, if the system had to use the representation shown in Figure 2.7 to check whether the move of the black queen to column 1, row 5 is valid, the information processor should check, among other things, that the representation square in column 2 and row 6 does not have any icon over it. Neither "representation square" nor "icon" is a relevant concept in the domain. It is much better that the information processor may ask questions like "Is the board square of column 6 and row 2 free?" where both *board square* and *free* are defined in the conceptual schema. For similar reasons, which will be explained, the information processor cannot use an internal representation.

What is most natural for the information processor is to use a representation based on the conceptual schema, which is the information base. However, the information base is virtual, because it does not exist physically within the system. When the information processor asks itself questions like "Is the board square in column 6 and row 2 free?" it behaves as if the information base really existed. In reality the question will be sent to the internal processor, which will answer it using the physical DB.

The representation of the state that the system has to maintain internally must allow, among other things, an efficient execution. That means the design of the internal representation must take into account technical factors. We call *internal schema* the representation form of the state of the domain used internally by the system, and *internal DB* the state representation in that schema. The internal DB is the only one that has a physical existence. The internal schema also includes the set of operations that can be invoked on the DB.

An internal schema for the system example that would be almost optimal from the point of view of the amount of space used (although not from other technical points of view) could be a file with the following record structure:

PieceType, Color, Row, Column

where PieceType could use one character (with a K for king, Q for queen, R for rook, etc.), Color one bit (0: white, 1: black), and Row and Column a single byte (number 1…8). Internal schemas, like the external ones, are defined with respect to the conceptual schema. In the example, the correspondence might be:

- The file has a record for each piece that is on the board.
- The first field indicates the piece type, the second its color, the third the row number of the board square where the piece is located, and the fourth the column number.

- The color of the board square is not represented explicitly. The external processor may infer it by adding the numbers of the row and the column: If the result is even, the board square is black; otherwise, it is white.

Using that internal schema, the partial contents of the internal DB corresponding to Figure 2.7 would be

R	1	8	2
R	1	8	4
K	1	8	7
Q	1	7	3
……			

The internal processor receives the commands issued by the information processor and executes them, possibly accessing the internal DB. For example, if the internal processor receives the command (or, as in this case, question) "Is the board square of column 6 and row 2 free?" it will check whether there is a record, in the above file, such that Row = 2 and Column = 6. If there is not such a record, the answer to the question will be positive, and negative otherwise. To perform its task, the internal processor needs to know the internal schema, including its correspondence with the conceptual schema.

Modern architectures of ISs are layered, with three logical layers: presentation, domain, and data management. The equivalent to the external processors is located in the presentation layer, the information processor in the domain layer, and the internal processor in the data management layer.

2.4 Requirements Engineering

Section 2.3 discussed the role of conceptual schemas in the architecture of ISs. Now, we are going to see their role in the development of the systems.

Conceptual schemas are the common base for external and internal schemas, as well as for their processors. Therefore, it is clear that it is not possible to design the architecture of an IS without the conceptual schema. Conceptual modeling must precede system design.

It is important to realize that it is impossible to design a system without knowing its conceptual schema. The only available options are either to define explicitly the schema or to have it in the minds of the designers. Unfortunately, sometimes the latter option is taken.

The stage that precedes system design is called *requirements engineering* [14]. Its objective is to capture the requirements that must be satisfied by the system. Normally, requirements engineering is a complex process, because the many persons (users, designers, managers, etc.) involved in it may have different views, needs, and interests.

Requirements engineering consists of three main phases, which can be performed iteratively:

- Requirements determination;
- Requirements specification;
- Requirements validation.

During *requirements determination*, the future users of the system and the designers analyze the problems, the needs, and the domain characteristics. On the basis of that analysis, they decide the changes to be introduced in the domain and the functions that should be performed by a new IS. Requirements determination is a crucial phase, because it determines a significant part of the final success or failure of the whole project. In this phase, it is decided how the future system will be, and an error in the decision often implies that users eventually will get an inadequate system.

During this phase, a conceptual schema of the existing domain may be elaborated, if it is considered necessary to achieve a common understanding of the domain. A conceptual schema of the desired domain can also be elaborated, without determining yet the part that will correspond to the new IS.

In the *requirements specification* phase, the functional and nonfunctional requirements of the new system are defined. The result is a set of documents (called specifications) that describe exactly the system that the users want and that the designers have to design and build. Functional requirements describe what the system must do, while nonfunctional requirements describe global properties of the system, like, for example, response time or portability.

The conceptual schema of an IS is the specification of the functional requirements of the system. The conceptual schema specifies all functions (memory, informative, and active) that must be performed by the system

and, together with the nonfunctional requirement specification, corresponds to the system specification.

During *requirements validation*, specifications are checked with respect to users' needs. In this phase, it must be ensured that users get a complete understanding of how the future system will be before it is built. This is also a crucial phase that can be done well only if requirements have been described explicitly.

Validation can be performed in two main ways:

- By presenting the conceptual schema and in general the specifications in a language and form that is easily understood by users. If the conceptual modeling language used is not completely understandable by the users, it will be necessary to provide either some help for its interpretation or translation to more familiar languages (not excluding natural language). When the conceptual schema is large, as is often the case, its structuring in fragments or views may be mandatory.
- By building (partial) prototypes of the system. If the conceptual modeling language used is formal, then prototypes may be generated automatically. This form of validation is usually more effective than the other form, but in general it is more expensive.

In summary, conceptual schemas are elaborated during the requirements engineering stage and are the basis for the next stage, system design.

For further details on how these activities can be facilitated by computers, see Chapter 13.

2.5 Desirable Properties of Conceptual Schemas

Now that we have seen what the conceptual schemas are and their role in the architecture of the system and during the development process, this section describes which properties should have these schemas in order to play those roles effectively [15–17].

A well-known property of conceptual schemas is the *100% principle*, or *completeness*, which states that

> All relevant general static and dynamic aspects, i.e., all rules, laws, etc., of the universe of discourse should be described in the conceptual

> schema. The information system cannot be held responsible for not meeting those described elsewhere, including in particular those in application programs [10].

The justification for the 100% principle is that a conceptual schema is the definition of the general domain knowledge the IS needs to perform its functions; therefore, the conceptual schema must include all required knowledge. If we had a "compiler" able to generate a system from the conceptual schema, then it would be obvious that the system could not contain anything not included in the schema. A conceptual schema is complete if it satisfies this property.

An important conclusion from the 100% principle is that the conceptual modeling language used must allow the description of all relevant aspects of a domain.

The *correctness* property is complementary to the completeness property: A conceptual schema is correct if the knowledge that defines it is true in the domain and relevant to the functions the IS must perform. For example, in our chess-playing system the fact that players have an address is probably irrelevant.

The Venn diagram in Figure 2.9 shows graphically the relationship between completeness and correctness. The left circle, A, represents the domain knowledge the IS needs to know to perform its functions. The right circle, C, represents the knowledge defined in the conceptual schema. In a complete conceptual schema, A is a subset of C. In a correct conceptual schema, C is a subset of A. In a complete and correct conceptual schema, A = C.

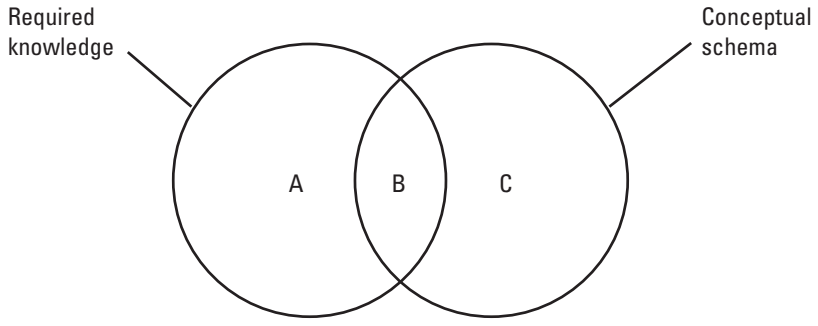

Figure 2.9 Completeness and correctness.

Correctness and completeness of a conceptual schema are checked during the requirements validation phase.

Another property that has become popular is the *principle of conceptualization*, which states that

> A conceptual model should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus excluding all aspects of (external or internal) data representation, physical data organization and access as well as aspects of particular external user representation such as message formats, data structures, etc. [10].

The justification is similar to the previous one: If a conceptual schema is the basis for system design, then it should not include any design aspect, thus leaving freedom to designers to decide on all those aspects. On the other hand, when a schema focuses only on conceptual aspects, it is simpler and, therefore, easier to be understood by users. A conceptual schema that satisfies this principle is called *design independent*.

Conceptual schemas are described in some conceptual modeling language. This language will have a set of rules that must be respected. A conceptual schema is *syntactically valid* (or just valid) if it respects all the rules of the language in which it is written. Syntactic correctness of a schema is independent of the domain.

Sometimes, the same piece of knowledge about a domain may be expressed in two or more ways in a given language. The property of *simplicity* states that simple schemas must be preferred, that is, schemas that use fewer language constructs or less complex constructs.

Closely related to the simplicity property is the property of *ease of understanding*. A conceptual schema should be easily understandable by the persons involved in the development of the IS, particularly its future users. Section 2.4 mentioned the importance of this property during requirements validation.

Finally, we mention the property of *stability*, also called flexibility, extensibility, or modifiability. A conceptual schema is stable if small changes in the properties of the domain or in the users' requirements do not imply large changes in the schema.

There are some proposals of metrics for evaluating these properties in a conceptual schema (see Chapter 14). A representative example is [18]. However, this is an issue where more work needs to be done to be fully practical.

References

[1] Langefors, B., "Information Systems," *Proc. IFIP '74*, North Holland, 1974, pp. 937–945.

[2] Checkland, P., *Systems Thinking, Systems Practice*, New York: Wiley, 1981.

[3] Boman, M., et al., *Conceptual Modelling*, New York: Prentice-Hall, 1997.

[4] Wand, Y., and R. Weber, "An Ontological Analysis of Some Fundamental Information Systems Concepts," *Proc. 9th. Intl. Conf. on Information Systems*, Minneapolis, MN, Dec. 1988, pp. 213–225.

[5] Motro, A., "Intensional Answers to Database Queries," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 6, No. 3, June 1994, pp. 444–454.

[6] Papazoglou, M. P., "Unraveling the Semantics of Conceptual Schemas," *Comm. ACM*, Vol. 38, No. 9, Sept. 1995, pp. 80–94.

[7] Ellis, J. R., *Objectifying Real-Time Systems,* New York: SIGS Books, 1994.

[8] Uschold, M., and M. Gruninger, "Ontologies: Principles, Methods, and Applications," *Knowledge Engineering Review*, Vol. 11, No. 2, 1996, pp. 93–136.

[9] Mylopoulos, J., "Information Modeling in the Time of the Revolution," *Information Systems*, Vol. 23, No. 3/4, 1998, pp. 127–155.

[10] ISO/TC97/SC5/WG3, *Concepts and Terminology for the Conceptual Schema and the Information Base,* J. J. Van Griethuysen (ed.), Mar. 1982.

[11] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach,* Englewood Cliffs, NJ: Prentice-Hall, 1995.

[12] Cook, S., and J. Daniels, *Designing Object Systems: Object-Oriented Modelling With Syntropy,* New York: Prentice-Hall, 1994.

[13] Motro, A., "Integrity = Validity + Completeness," *ACM Trans. Database Systems,* Vol. 14, No. 4, 1989, pp. 480–502.

[14] Loucopoulos, P., and V. Karakostas, *System Requirements Engineering*, New York: McGraw-Hill, 1995.

[15] Bubenko, J. A., Jr., "Validity and Verification Aspects of Information Modeling," *Third Intl. Conf. on VLDB*, Tokyo, Oct. 1977, pp. 556–565.

[16] Davis, A. M., *Software Requirements: Objects, Functions, and States,* Englewood Cliffs, NJ: Prentice-Hall, 1993.

[17] Lindland, O. I., G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modeling," *IEEE Software*, Mar. 1994, pp. 42–49.

[18] Moody, D. L., "Metrics for Evaluating the Quality of Entity Relationship Models," *Proc. 17th Intl. Conf. on Conceptual Modeling*, Singapore, Nov. 1998, LNCS 1507, Springer, pp. 211–225.

Selected Bibliography

Batini, C., S. Ceri, and S. B. Navathe, *Conceptual Database Design: An Entity-Relationship Approach*, Redwood City, CA: Benjamin/Cummings, 1992.

This book is devoted to conceptual modeling but focuses on DBs.

Nijssen, G. M., and T. A. Halpin, *Conceptual Schema and Relational Database Design*, New York: Prentice-Hall, 1989.

Chapter 2 of this book (along with Chapter 4 of [14]) is an appropriate general introduction to conceptual modeling.

Borgida, A., S. Greenspan, and J. Mylopoulos, "Knowledge Representation as the Basis for Requirements Specifications," *IEEE Computer*, Apr. 1985, pp. 82–91.

This article emphasizes principles with reference to languages.

Loucopoulos, P., "Conceptual Modeling," in *Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development,* P. Loucopoulos and R. Zicari (eds.), New York: Wiley, 1992, pp. 1–26, and Rolland, C., and C. Cauvet, "Trends and Perspectives in Conceptual Modeling," pp. 27–48 in the same book, provide a complete picture of conceptual models and conceptual modeling languages, including many references.

Falkenberg et al., "A Framework of Information System Concepts: The FRISCO Report," IFIP WG 8.1 Task Group FRISCO, Dec. 1996.

This report is a recent in-depth treatment of IS concepts.

Boman, M., et al., *Conceptual Modeling,* Upper Saddle River, NJ: Prentice-Hall, 1997.

This is one of the very few books that deals entirely with conceptual modeling of information systems.

Mylopolous, J., "Information Modeling in the Time of the Revolution," *Information Systems,* Vol. 23, No. 3/4, 1998, pp. 127–155.

This article gives a modern view of the field.