2

Discrete Dynamic Systems

This chapter introduces discrete dynamic systems by first looking at models for dynamic and static aspects of systems, before covering continuous and discrete systems. Transition systems are discussed and formally defined, since these play an important role in behavioural modelling. Causality and events are covered, because these concepts form the basis of system behaviour. An application scenario is introduced to illustrate the concepts covered in this chapter.

2.1 Characterisation of Dynamic Systems

Every system that we build shows some behaviour, and this is of a dynamic nature. This section introduces a taxonomy of system models to represent dynamic systems. The taxonomy is organised in three dimensions. The first dimension organises models regarding the static and dynamic aspects represented. The second dimension refers to the granularity of the state changes of a system, which can happen either continuously or in discrete steps. The final dimension organises models in those that represent sequential behaviour and those that allow us to model concurrent behaviour of dynamic systems.

2.1.1 System Models for Static and Dynamic Aspects

Software systems are complex entities that can only be described by looking at different aspects in turn, using different models. We choose the type of model depending on the particular modelling goal. If we are interested in the structural aspects of a system, we use different modelling languages from what we use for modelling dynamic aspects. Rather than talking about dynamic and static systems, we discuss dynamic and static aspects of a system, using the appropriate modelling languages.

In the previous chapter we discussed two modelling languages that can be used to represent the dynamic and static properties of a system, i.e., data

M. Kunze and M. Weske, *Behavioural Models*,
DOI 10.1007/978-3-319-44960-9_2

models and state diagrams. With state transitions, we can model dynamic aspects and with data models static ones.

Data models describe a static aspect of a system, because they provide a blueprint of the data stored in the system. In the example shown in Fig. 1.6, each customer has a customer identification, a name, and an address, while each order has an order identification, data, and an amount. Data models specify the structure of the data processed by systems; they do not make any stipulations about the behaviour of the system, i.e., about dynamic aspects, respectively.

For instance, data models cannot be used to specify that a customer can put articles in a shopping basket only after the customer has been authenticated. Causal dependencies cannot be expressed in data models, nor in other types of static models, such as software architectures. Therefore, data models and also software architectures provide a means to express static aspects of systems.

In contrast, state diagrams are dynamic models, since they explicitly consider states and the state transitions that a system can perform. In case the of the online shop, the state diagram explicitly introduced in Fig. 1.7 represents the states of the shop and the state transitions that are possible. We can use state diagrams to impose constraints on the behaviour of systems, for instance by allowing an order to be paid for only after shopping has been completed. Since state diagrams allow us to express constraints on system behaviour, they describe dynamic aspects of systems.

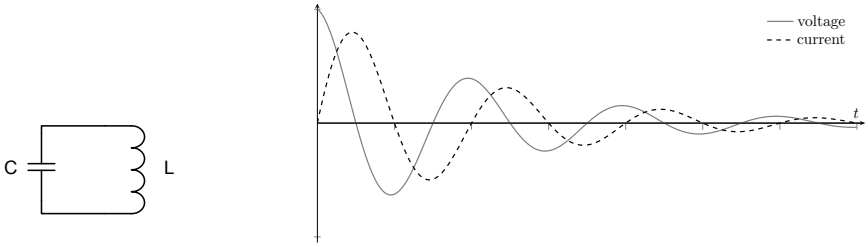
2.1.2 Continuous and Discrete Systems

Computer systems are discrete systems, where a state is specified by a collection of binary values. But there might be system components that are of continuous nature, for instance components containing sensors that provide continuous values. Dynamic systems can be organised into continuous and discrete systems.

In a continuous system model, states are represented by a continuous domain and state transitions are specified by a continuous function, whereas state changes occur in steps in discrete dynamic systems.

An example of a continuous system is an inductor–capacitor circuit, which, in the simplest case, is an electric circuit consisting of an inductor and a capacitor. It is used, for instance, to isolate frequencies in complex signals. Figure 2.1a shows a schematic view of such a circuit, where L stands for the inductor and C for the capacitor.

Without going into the technical details, the LC circuit works as follows. If a charged capacitor is connected to an inductor, an electric current will flow through the inductor, thereby discharging the capacitor, which creates a magnetic field in the inductor. Once the capacitor is discharged, the magnetic field induced in the inductor causes an electric current to flow in the circuit in the same direction. This results in the capacitor being charged again, this time in reverse polarity. Then the process starts all over again.



(a) LC circuit (b) Variation of voltage and electric current with time

Fig. 2.1: System model of an inductor–capacitor circuit

The LC circuit is a continuous system. The state of the system, i.e., its voltage and current, consists of continuous dimensions. The state transitions are continuous functions that change the state over time, as depicted in Fig. 2.1b. As the capacitor is charged with the opposite polarity by the induced magnetic field in the inductor, the voltage and current alternate between positive and negative values.

This diagram shows that the voltage is highest when the capacitor is maximally charged and no electric current is flowing through the circuit, whereas the current is maximum when the capacitor is discharged and the voltage is zero. Owing to electrical resistance in the circuit, the maximum values of the current and voltage slowly decrease.

The majority of models in computer science, however, are not concerned with continuous behaviour, as one is interested in the operations that a system performs to achieve its goal. Examples are the operations executed by an algorithm and the activities carried out to decide whether a credit application should be granted or not.

For this purpose, the continuous time dimension is discretised. That is, it is split into intervals and the system is described only at the expiration of each interval. As a consequence, the states of a system can be represented by a set instead of by a continuous dimension. Depending on the modelling goal, a state may include an identity, a textual description, or a composite model itself, for instance a data model. Then, the state transitions form a temporal relation between states.

This is in line with the physical construction of a computer. In the electrical circuits of a CPU, low and high voltages denote values of 0 and 1, respectively. As the voltage is a continuous function of time, a clock is introduced and voltages are interpreted only at clock ticks. Discrete dynamic systems adopt this concept by imposing a discrete time model. At any point in time, the system is in one particular state, whereas the state of the system may change when a clock tick occurs.

As mentioned above, computer systems are of a discrete nature. We have witnessed this property in the online store example and its state diagram, which consists of a set of discrete states and transitions between them.

Since discrete dynamic system models are at the centre of this book, the key properties of these models will be illustrated by an additional example, shown in Fig. 2.2. This figure shows the states of a machine that produces metal casings. Regarding the modelling language, we are using the same conceptual model as we have already used in the first version of the online shop example. However, there is a slight difference in the notation: states are represented by circles, not by ellipses. We will use this notation again later in this book to express different types of state diagrams.

In Fig. 2.2, circles represent the states of the machine; state transitions are shown by directed arcs. The arrowhead of an arc denotes the target state that can be reached from the source state through a state transition. The machine starts when a piece of metal is available as raw material for the production of a casing. The piece is first cut into a rectangular shape, which is then bent into a cylinder. The edges are welded and, finally, the casing is cooled. The model abstracts away the actual time needed to produce the casings and represents only the ordering of states.

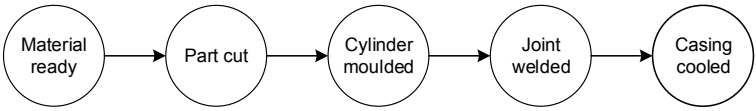


Fig. 2.2: Model of a machine that produces metal casings

The above example describes a machine that performs production steps, without investigating the nature of the state transitions. A variant of the diagram is shown in Fig. 2.3, in which each state transition is marked with a label that describes an action carried out during production. This model is richer, since it identifies the actions required for each state transition.

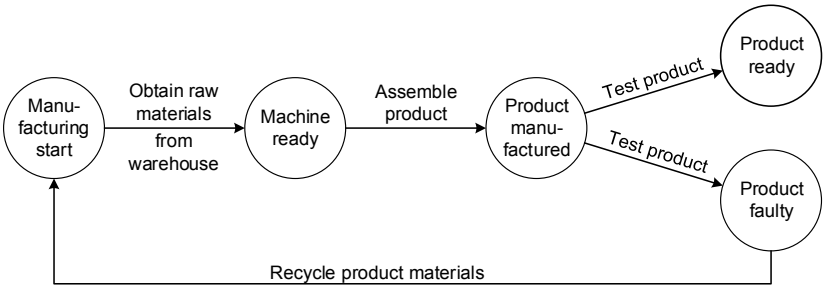


Fig. 2.3: Model of a manufacturing process

When production starts, raw material needs to be obtained from the warehouse and placed into the machine. Once the machine is ready, the product can be assembled. When the manufacture of a product is completed, the product is tested. The test can result in two different states: either the test succeeds and the product is ready for shipment, or the product fails the test and must be recycled, and production starts again.

2.1.3 Sequential and Concurrent System Models

The systems discussed so far are of a sequential nature. In the online shop, a customer first enters the shop and selects products, before submitting payment information. Finally, the products are shipped and the process completes. Sequential behaviour can be characterised by a sequence of state transitions. The sequence of states of the online shop can be seen in the state diagram shown in Fig. 1.2.

Sequential behaviour can also include choices. In a state diagram, a choice is represented by a state with multiple transitions. We have seen this type of state diagram in the manufacturing machine example. Figure 2.3 shows this situation because, after the product has been manufactured, either the state *Product ready* or the state *Product faulty* is reached. In either case, the behaviour of the system is sequential. Models to capture the behaviour of sequential systems will be covered in Chapter 3.

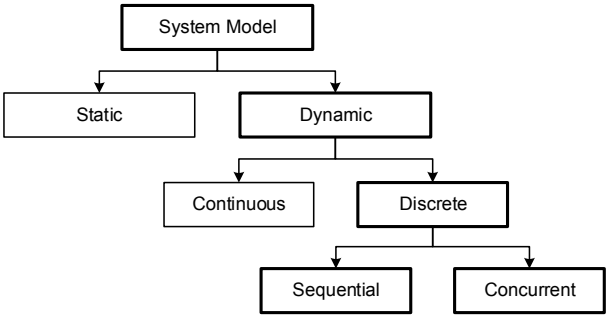


Fig. 2.4: Classification of systems and system models

Whenever a system contains several subsystems that can proceed independently of each other, sequential models are not adequate for capturing the behaviour of the system. The behaviour of these types of systems is called concurrent, because there are subsystems that are active and can proceed at the same time, concurrently. Specific modelling concepts and languages have been introduced to represent concurrent systems. These concepts and languages will be covered in Chapter 4.

A categorisation of systems and their respective system models is shown in Fig. 2.4. The top part of that figure shows a broad classification of system models into static and dynamic ones. The respective models have been sketched in this section.

Some key properties of sequential and concurrent systems have also been touched upon. A complete picture of system models will be provided in Chapters 3 and 4, when sequential and concurrent system models are investigated.

2.2 Transition Systems

In order to reason about the behaviour defined by discrete dynamic system models, state transition systems need to be formally defined.

2.2.1 State Transition System

A state transition system was informally introduced in the previous chapter to describe the behaviour of an online shop, as shown in Fig. 1.2. A state transition system consists of a set of states and a state transition relation. It is formally defined as follows.

Definition 2.1 A *state transition system* is a pair (S, δ), where

- S is a finite set of states,
- $\delta \subseteq S \times S$ is a state transition relation. ◇

The set of states and the state transition relation can be visualised by a graph representation, which consists of nodes and edges. Each state $s \in S$ is represented by a node, and each state transition $(s, s') \in \delta$ by an edge between states s and s'. Since (s, s') is an ordered pair, the edge is directed from s to s'.

From Definition 2.1, it follows that two nodes can be connected by at most one edge in the same direction.

To illustrate the concept of a state transition system, we revisit the behavioural model of the online shop shown in Fig. 1.2. For convenience, we abbreviate the states as follows: *Shop entered (se), Shopping completed (sc), Order paid (op)*, and *Products shipped (ps)*. Formally, this state transition system can be described by (S, δ), such that

$$S = \{se, sc, op, ps\}.$$

The four states, represented by ellipses in the graphical representation of the state transition system, are reflected by the four elements in the set of states. With these states, the transition relation shown in Fig. 1.2 is defined by

$$\delta = \{(se, sc), (sc, op), (op, ps)\}$$

The three arcs shown in the graphical representation of the state transition system in Fig. 1.2 are reflected by the three tuples in the transition relation.

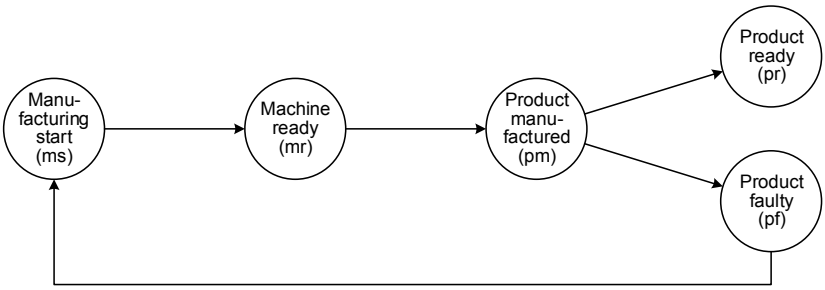


Fig. 2.5: State transition system of production machine

To discuss the semantics of state transition diagrams in more detail, a variation of the production process introduced earlier is shown in Fig. 2.5. The variant used here has the same states as in the one discussed earlier, but the state transitions are not labelled.

This state transition system consists of five states with their respective abbreviations: *Manufacturing start (ms), Machine ready (mr), Product manufactured (pm), Product ready (pr)*, and *Product faulty (pf)*. (S, δ) of this state transition diagram are defined as follows:

$$S = \{ms, mr, pm, pr, pf\},$$
$$\delta = \{(ms, mr), (mr, pm), (pm, pr), (pm, pf), (pf, ms)\}.$$

At each point in time, the state transition system is in exactly one state s, from which it may transition to another state s' if there exists a state transition $(s, s') \in \delta$.

If more than one state transition is possible for a given state s, for instance, if there are two state transitions $(s, s'), (s, s'') \in \delta$ such that $s' \neq s''$, then s' and $s''$are exclusive, i.e., exactly one state transition is chosen.

In the example, there are two transitions possible in the state *Product manufactured*. Each time this state is reached, one of the transitions is used and the other is discarded.

For instance, if in the first iteration the product is faulty, then the lower transition to the state *Product faulty* is chosen and manufacturing is started again. If the second iteration results in a product that meets its specification, the upper transition to the state *Product ready* is chosen.

2.2.2 Labelled State Transition System

So far the behaviour of discrete dynamic systems has been represented by states and state transitions. The state transition diagram of the online shop, for instance, had several states and transitions between them. However, we did not represent any actions in the system that actually triggered these transitions. In this section, we present not only the modelling of the states and state transitions of a system, but also that of the actions that trigger these transitions.

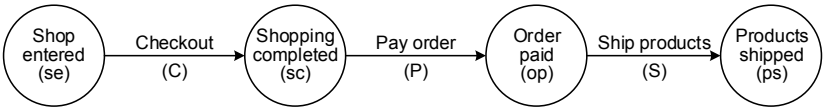


Fig. 2.6: States and labelled state transitions for an online shop

Figure 2.6 shows a variant of the state transition diagram for the online shop that captures these actions. This version of the model is much richer, since it associates actions that the system takes with state transitions. Given this model, system designers have to make sure that the shop transits from the state *Shop entered* to the state *Shopping completed* only by an action called *Checkout*.

The term "action" refers to any function or procedure that the system executes to perform a state change. The abstraction property of models also applies to state transitions. It is a modelling decision which functionality is associated with a particular action. Therefore, the term "action" subsumes function calls, procedure calls, and other types of events that may happen.

The model does not define any further constraints on how the checkout action should be implemented. In the example of the online shop, this action is typically implemented by a button in the user interface of the online shop. When that button is clicked on, a procedure is called on the server side, which sends a web page to the browser with the shopping cart and further information, for example, the overall amount payable.

Models like the one shown in Fig. 2.6 are called labelled state transition systems. They are very useful during system design, because they constrain the execution of procedures to certain states of the system. For instance, invoking the checkout action in the state *Order paid* would not make sense, and therefore this action is disallowed in the labelled state transition diagram shown in Fig. 2.6.

On the other hand, certain sequences of actions by the system are allowed in the labelled state transition diagram. In the case of the online shop, the only sequence that is allowed is *Checkout*, *Pay order*, *Ship products*. When we discuss the behaviour of systems using labelled state transition diagrams, such actions are typically referred to as "input". So we can say that the online shop

can reach the state *Shopping completed* from the state *Shop entered* with the input *Checkout*.

To generalise, each state transition is associated with some input, and different inputs are represented by different symbols. The set of all input symbols of a transition system is called its *alphabet*, denoted by Σ. Generally speaking, the alphabet consists of all labels in the state transition system. Each of these labels refers to exactly one action that can be performed by the system.

Definition 2.2 An *alphabet* is a finite, non-empty set of symbols, denoted by Σ. The symbols in Σ refer to actions that a system can perform. ◇

Owing to the abstraction that models provide, the software system can perform many more actions. However, the alphabet of a labelled state transition system represents all of the actions of a software system that are relevant for the modelling purpose. In this book, we consider finite alphabets only, as we are concerned with the modelling of systems for which the set of possible actions that the system can perform is known a priori.

Definition 2.3 A *labelled state transition system* is a tuple (S, Σ, δ) such that

- S is a finite set of states,
- Σ is a finite alphabet, and
- $\delta \subseteq S \times \Sigma \times S$ is a state transition relation. ◇

In a labelled state transition system, each state transition $(s, l, s') \in \delta$ consists of a source state s and a target state s' and is assigned a label $l \in \Sigma$. The graphical representation of a labelled transition system is much like that of a state transition system. Each tuple $(s, l, s') \in \delta$ is represented by a directed arc from s to s', which is labelled by l.

By using the abbreviations for the state and transition labels, we can express the labelled state transition diagram shown in Fig. 2.6 as

$$S = \{se, sc, op, ps\},$$
$$\Sigma = \{C, P, S\},$$
$$\delta = \{(se, C, sc), (sc, P, op), (op, S, ps)\}.$$

The elements of the state transition relation are no longer pairs of states but triples, consisting of the source state, transition label, and target state.

The introduction of labels also allows us to define multiple transitions between a given pair of states. As a consequence, there may be several edges leading from one state to another in the graphical representation of the system model, each of which has a different label.

The semantics of such a situation is that the system model allows different state transitions from a state s to a state s'. Since the labels of the state transitions are different, the actions performed by the system are different, too. However, their effects on the modelled state change of the system are identical.

This is the result of abstraction during the modelling process. In the real system a different action is executed, which might also have slightly different effects on the state of the system, but this difference is not relevant to the model and therefore is abstracted away.

An example of such a situation is given in Fig. 2.7a, where there are different ways of sending a quote. The model expresses these different ways (sending the quote by either email, fax, or letter), showing the system engineers that several alternative ways of sending the quote need to be developed. However, after sending the quote using any of these ways, the system is in the state *Quote sent*.

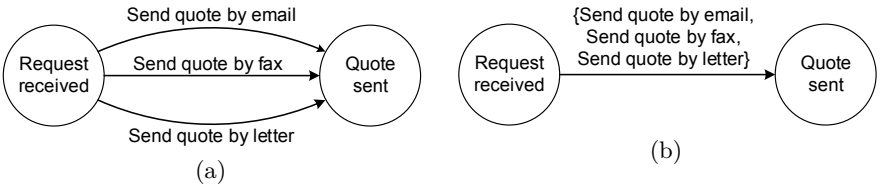


Fig. 2.7: Two labelled transition systems for a quote management system that have several state transitions between a given pair of states

To reduce the visual complexity of a labelled state transition system, state transitions that share the same source and target states can written in a compact way using a single arc. This is shown for the above example in Fig. 2.7b. The label of the arc lists the set of inputs of these state transitions. This is a visual simplification but has no impact on the state transition relation. Basically, two slightly different notations have been used to express the same set of concepts.

2.2.3 Sequences of Inputs

Traversing the state transitions of a labelled transition system leads to sequences of symbols. Assuming that each labelled state transition represents an operation or activity carried out by the system, the sequences of a transition system denote a set of valid actions and the order in which they can be carried out. A sequence can, therefore, be interpreted as one possible behaviour of the system modelled.

The set of all sequences that a labelled transition system can generate characterises the set of all possible behaviours of that system. We formalise these sequences based on the alphabet of labelled transition systems.

Definition 2.4 For an alphabet Σ, we define sequences as follows.

- A sequence σ of length $n \in \mathbb{N}$ is defined by a function $\sigma : \{1, \ldots, n\} \mapsto \Sigma$ that assigns a symbol to each position in the sequence. We refer to the ith position in the sequence by $\sigma(i)$, $1 \leq i \leq n$; $|\sigma| = n$ is the cardinality of σ.
- The set of all sequences of length $n \in \mathbb{N}$ is denoted by Σ^n; ε denotes the empty sequence, such that $|\varepsilon| = 0$.
- Σ^* is the set of all sequences of finite length, i.e., $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$. $\diamond$

A sequence is an ordered list of symbols from the alphabet. For a sequence σ with $\sigma(1) = A$, $\sigma(2) = B$, and $\sigma(3) = C$, we can also write $\langle A, B, C \rangle$.

We shall now illustrate this concept using the online shop example. Figure 2.8 shows a new version of the labelled state transition diagram, which allows customers to resume shopping after checkout. This new behaviour is represented by a state transition from the state *Shopping completed* to the state *Shop entered*, which is labelled *Resume shopping (R)*.

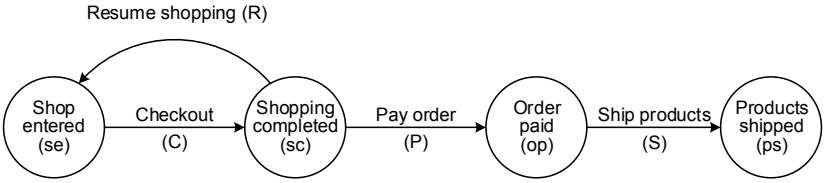


Fig. 2.8: Online shop labelled state transition system with loop

The labelled state transition diagram shown in this figure defines several sequences. In fact, owing to the cycle in the diagram, an infinite number of different behaviours are specified by the diagram. Consider a simple case, in which a user enters the shop, proceeds to checkout, and pays for the order, before the products are shipped. This behaviour is represented by the sequence

$$\langle C, P, S \rangle.$$

If the customer chooses instead to resume shopping after the first checkout, the following sequence occurs:

$$\langle C, R, C, P, S \rangle.$$

Intuitively, these sequences start at the beginning of the process and end when the shopping process is completed. However, sequences are not limited to

this property. Any partial behaviour of the system can be represented by a sequence; hence, the following sequences are also valid for the state transition diagram shown in Fig. 2.8:

$$\langle P, S\rangle, \langle R, C, P\rangle, \langle S\rangle, \langle R, C, R, C, R, C, R, C, P\rangle.$$

In contrast, the following sequences do not represent valid behaviour of the online shop:

$$\langle S, P\rangle, \langle R, S, C, P\rangle, \langle S, C\rangle, \langle R, R\rangle.$$

We can formalise these considerations in the following definition.

Definition 2.5 Let (S, Σ, δ) be a labelled state transition system. The *state transition relation for sequences* $\delta^* \subseteq S \times \Sigma^* \times S$ is defined as follows.

- $(s_1, \sigma, s_k) \in \delta^*$ holds if $\sigma = \langle l_1, \ldots, l_{k-1}\rangle$ and $(s_i, l_i, s_{i+1}) \in \delta$ for $1 \leq i < k$.
- The empty sequence does not advance the state of the system:

$$(s, \varepsilon, s') \in \delta^* \implies s = s'$$

◇

With this definition, we can show that $\langle R, C, R, C, R, C, R, C, P\rangle$ is a correct sequence by providing the states before the sequence is traversed (in this case the state *sc*) and the state after sequence traversal, *op*. It holds that

$$(sc, \langle R, C, R, C, R, C, P\rangle, op) \in \delta^*$$

because

$$(sc, R, se), (se, C, sc), (sc, R, se), (se, C, sc),$$
$$(sc, R, se), (se, C, sc), (sc, P, op) \in \delta.$$

This shows that for each label in the sequence, there is a corresponding state transition in the state transition system that originates from the current state. Owing to the loop in the labelled state transition system, it would of course suffice to show that

$$(sc, R, se), (se, C, sc), (sc, P, op) \in \delta.$$

2.3 Events and Causality

In this section we provide an abstract view of discrete dynamic systems, which serves as a foundation for the remainder of this book. This view is based on two essential concepts for specifying the behaviour of dynamic systems: events and causality.

An event is something that happens, a concrete occurrence in a particular system or its environment. Events happen in the real world, such as receiving a parcel, sending a letter, or inserting a 50 ct coin into a ticket vending machine. Events also happen in software systems, such as pressing the checkout button in an online shop, submitting a form with payment details, or invoking a procedure.

Behavioural models use events and provide an ordering between events or, more generally, define execution constraints between events. The reason for these execution constraints is causality.

We can illustrate the concepts of events and causality by applying them to the online shop example shown in Fig. 2.8. The labelled state transition system captures the behaviour of the online shop. The online shop has four states and three state transitions. State transitions represent events. When the system is in the state *Shop entered*, the checkout event C can occur. However, the payment of the order event P can only occur after the checkout event has occurred.

The reason for this execution ordering of events is *causality*, and the effect is *causal ordering*. Two events are causally ordered if the occurrence of one event depends on the occurrence of the other event beforehand. In the example, we can pay for the order (event P) only after checkout has occurred (event C). Therefore, the event P is causally dependent on C. There is no way that the system can allow P to happen if C has not happened before it.

By defining causality between events, models can specify the behaviour of dynamic systems. We have discussed causality for labelled state transition systems, but in the remainder of this book, several techniques for expressing causality of events in a dynamic system will be covered.

The causality defined by the state transition system shown in Fig. 2.8 can be uncovered by looking at the execution sequence

$$\langle C, R, C, P, S \rangle.$$

This execution sequence starts with checkout, C, followed by an event that indicates resumption of shopping, R. Since R is causally dependent on C and C has already occurred, this sequence of events is a valid behaviour of the system, as specified by the labelled state transition system. Executing $\langle C, R \rangle$ brings us back to the state *Shop entered*. Now, again, checkout has to occur, before a payment event P can occur and the products are shipped as indicated by the event S.

2.4 Application of Discrete Dynamic System Models

This section uses an example to investigate the mapping, abstraction, and pragmatics properties of behavioural models that specify discrete dynamic systems.

The basis of this summarising example is the online shop shown in Fig. 2.6. That diagram focused on the main phases of the online shop, disregarding essential functionality such as authentication of users.

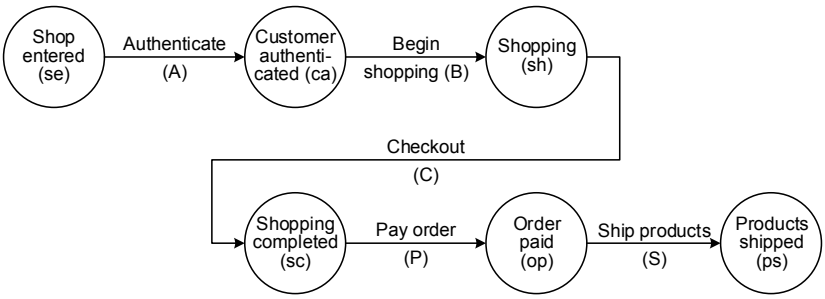


Fig. 2.9: Online shop labelled state transition system with authentication

The state transition diagram in Fig. 2.9 represents an updated version of the online shop which includes authentication. After entering the online shop, customers have to authenticate first before they begin shopping. If they have already registered at the online shop, they can use their credentials to log on to the system.

If this is not the case, they first have to register. In any case, a customer can start to fill their shopping cart only after they have been authenticated. This makes sure that every shopping cart can be associated with an authenticated customer.

When the customer has selected the desired products, they can proceed to checkout by selecting the checkout button on the web page of the online shop. Thereby the *Checkout* state transition is performed, and the system changes its state to *Shopping completed*.

Both states involved are abstractions of the original, i.e., the software system that implements the online shop. But the *Checkout* state transition is also an abstraction, because it represents an action in the software system that triggers the state change.

Technically, many operations are executed to achieve this state change. After the user has clicked on the appropriate button, the client sends an HTTP request to the web server of the online store. On receiving this request, the web server invokes a procedure in the application server that finally invokes a stored procedure in the database system to store the information persistently, including the contents of the customer's shopping cart.

This discussion shows the abstraction that takes place regarding state transitions in behavioural models.

The mapping property of models also holds, because the selection of the checkout button leads to a state change in the system, which is represented in the model. This state change is also reflected on the client side, i.e., the

web page that is sent to the customer's web browser. This page shows the contents of the shopping cart and associated information, for example, the amount payable. The state transition in the model can be mapped to a series of steps that are taken by the system to complete shopping.

The model also satisfies the pragmatics property, because it can serve as a replacement for the software system. We can discuss the behaviour of the system and the states that are reachable from a given state without being distracted by details of the technical implementation. The pragmatics property is important when it comes to new requirements that have to be incorporated into a software system.

Based on the version of the online shop specified in Fig. 2.9, we now assume that new requirements emerge.

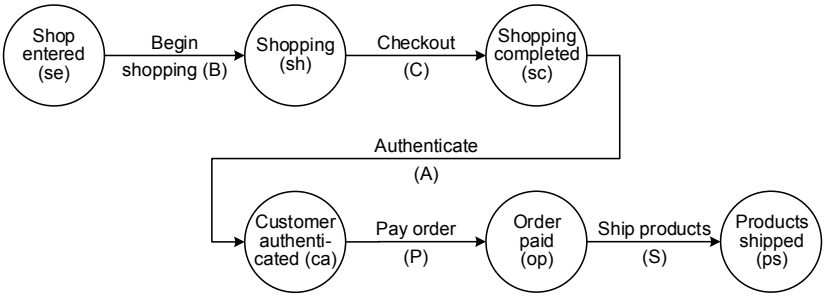


Fig. 2.10: Labelled state transition system for online shop after new requirements have been incorporated

By analysing web logs, the management team finds out that in many cases customers have left the shop without authenticating. The inference is that customers do not want to authenticate before they browse the shop and find products that they like. Clearly, before paying for their order, customers need to have authenticated, but they do not need to do so before then. This leads to a the new requirement to delay authentication until the latest possible point in time, i.e., immediately before payment.

The labelled state transition system proves useful in redesigning the online shop accordingly. The management team decides to shift the state *Customer authenticated* to after the state *Shopping completed* but before the state *Order paid*. This allows customers to enter the shop and immediately start shopping, before being asked for authentication. The resulting extended version of the online shop is shown in Fig. 2.10.

Bibliographical Notes

The fundamentals of modelling, including the model characteristics discussed in this chapter, were introduced in a seminal book by Stachowiak (1973). There are several books on modelling in computer science. Henderson-Sellers (2012) looks at modelling from a mathematical and ontological perspective. Different aspects related to the syntax and the semantics of models in computer science are discussed by Harel and Rumpe (2004).

In a book edited by Embly and Thalheim (2011), different types of conceptual models are discussed, ranging from data models to interaction modelling and modelling of systems requirements. The representation and analysis of business processes are the central features of a textbook by Weske (2012), which introduces the concepts of process modelling and a set of languages to represent business processes.

http://www.springer.com/978-3-319-44958-6

Behavioural Models
From Modelling Finite Automata to Analysing Business Processes
Kunze, M.; Weske, M.
2016, XII, 279 p. 177 illus., Hardcover
ISBN: 978-3-319-44958-6