Chapter 9
The DELTA Theory: Understanding Discrete Event Systems

Abstract The DELTA theory, also called EE system theory, is a theory about the construction and operation of systems in general. The realm of systems is divided into three regions: organised simplicity, organised complexity, and unorganised complexity. The definition of a (homogeneous) system is presented as a triple $(\mathbb{C}, \mathbb{E}, \mathbb{S})$, where $\mathbb{C}$ (composition) is a set of elements of some category, $\mathbb{E}$ (environment) is a set of elements of the same category as the elements in $\mathbb{C}$, and $\mathbb{S}$ (structure) is a set of interaction bonds among the elements in $\mathbb{C}$ and between them and the elements in $\mathbb{E}$. Examples of categories are: physical, biological, and social. Organisations belong to the category of social systems. Three sorts of conceptual models are distinguished: black boxes, grey boxes, and white boxes. The well-known finite automaton or finite state machine, and the discrete event system are examples of grey boxes. For a thorough discussion of the grey box and the white box, the PRISMA model is introduced. In this meta model, systems are considered to be discrete event automata, operating in a linear time dimension. Its formalised ontological model is particularly suited to study organisations. In the PRISMA grey box, three ways of mutual influencing between (the elements of) systems are distinguished, called activating, restricting, and impeding. The PRISMA white box allows one to conceive organisations as prismanets: networks of processors, channels, and banks. Prismanets are comprehensive formalised systems, open to formal analysis and to implementation in software. They can conveniently be expressed in prismanet diagrams. To illustrate the PRISMA model, two example prismanets are presented: one regards a traffic control system, and the other a car rental organisation. Next, the generic transaction prismanet is discussed. It is the understanding of the complete transaction pattern from the PSI theory in the PRISMA model. Lastly, the quality aspects of PRISMA models are discussed, as well as the importance of the PRISMA model for software engineering.

9.1 Introduction

The theory in this chapter is labeled Δ-theory. The Greek capital letter is pronounced as DELTA, which is an acronym for Discrete Event in Linear Time Automaton. It is a theory about the construction and operation of systems, in particular of discrete

J. L. G. Dietz, J. B. F. Mulder, *Enterprise Ontology*, The Enterprise Engineering Series, https://doi.org/10.1007/978-3-030-38854-6_9

event systems. In Chap. 4, the DELTA theory is classified as an ontological theory, meaning that it is about the nature of things. It serves foremost as a solid foundation for the other theories in this category: the PSI theory (Chap. 8), the ALPHA theory (Chap. 11) and the OMEGA theory (Chap. 10). In addition, the DELTA theory offers three sorts of meta models for studying systems: the black-box model, the grey-box model, and the white-box model.

Section 9.2 (foundations) consists of three subsections. Section 9.2.1 provides an introduction in systems theory and in systems thinking, including the ontological system concept that is adopted in this book. In Sect. 9.2.2, the three basic sorts of conceptual models are presented and discussed. The most primitive one is the black-box model; it doesn't contain any knowledge about the system's construction and the operation. This property makes it only suitable for studying possible function (s) and (external) behaviour. In contrast, the white-box model of a system contains all knowledge about its construction. It serves to study the construction of a system (i.e. the constituting parts and their interactions) and its operation (i.e. the effects of the interaction in the course of time). The grey-box model is a black-box model, but with an internal state. For the class of discrete event systems, a specific (white-box) meta model is presented and discussed in Sect. 9.2.3, called the PRISMA model. It allows one to build comprehensive, coherent, consistent, and concise white-box models, fully abstracted from realisation and implementation. These models are called essential prismanets. Their corresponding grey-box models are fully formalised, and therefore suited for formal analysis and for (discrete event) simulation. The PRISMA white-box and grey-box model is illustrated by a technical system (traffic control) and a social system (a part of a car rental company).

Section 9.3 (elaborations) starts with the presentation and discussion of the generic transaction prismanet, which is the expression of the complete transaction pattern (cf. Chap. 8) in the PRISMA model. Next, the quality aspects of PRISMA models are discussed. The section ends with a discussion of the implications of the PRISMA model for the field of software engineering. Section 9.4 (discussions) contains a comparison of the prismanet and the Petri net.

9.2 Foundations

9.2.1 Systems Thinking

9.2.1.1 Introduction

Systems thinking is an approach to problem solving which goes hence and forth between a global, holistic view on a system, and a detailed, specific view on its constituting parts. It originates from several areas, including General Systems Theory [1], Cybernetics [2], and System Dynamics [3]. Unfortunately, the practice of systems thinking suffers often from a lack of precision, notably regarding the notion of system itself. Instead of precise definitions, many textbooks only provide characterisations, such as "A system is a set of related elements with some purpose", and "The whole is greater than the sum of its parts". Taking the first one, our first

comment is that, according to the TAO theory (cf. Chap. 7), systems don't have purposes; only human beings do. The second comment is that proponents of this statement fail to separate the function and the construction perspective on systems. The second assertion (the whole is more than its parts) points to the distinctive property of systems as opposed to aggregates, but it has to be made more precise. In order to be called a *system*, its elements must act upon each other, in such a way that the trajectories, or processes that they cause to happen, are dependent on the mutual influencing of the elements, that is, that they are different from what these processes would have been if the causing elements would not interact. Or, as Bunge puts it, the assertion is a fuzzy version of the insight that ". . . the components of a concrete system are linked, whence the history of the whole differs from the union of the histories of its parts" [4] (Chap. 1). If the relationships between the elements are only passive, the thing is not a system but an *aggregate*. A well-known example of something that is an aggregate, but often called a system, is the Periodic Table of Mendeleev.

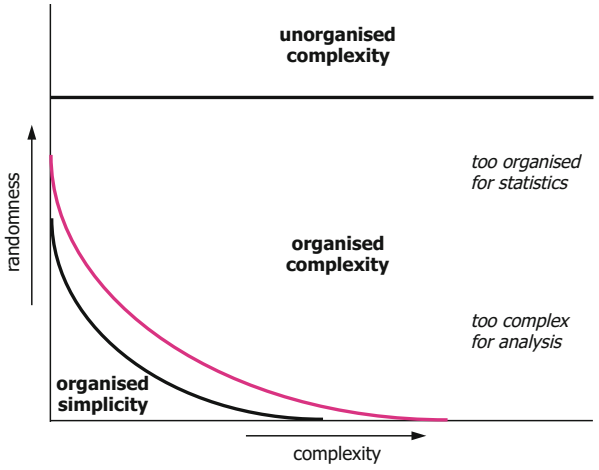

Fig. 9.1 Regions of systems with respect to methods of thinking

Weinberg [5] divides the realm of systems into three regions: organised simplicity, organised complexity, and unorganised complexity. An adapted version of his figure 1.9 is presented as Fig. 9.1. The region of *organised simplicity* comprises systems that have relatively few elements and mostly a great deal of structure. Systems in this region can generally be studied by (mathematical and logical) *analysis*. Examples are machines and other technical systems. The region of *unorganised complexity* comprises systems that have a very large number of elements with mostly few structural relationships. Because of the high level of randomness, systems in this region can generally be studied by *statistics*. Examples are populations of animals or plants, and vessels of gas molecules. In between these two is the region of *organised complexity*. It comprises systems that are too organised for statistics and too complex to be studied by analytical methods. This region is the core of enterprise engineering (EE): all enterprises belong to it. It is the ambition of the Ciao! Network to shift the border between organised simplicity and organised complexity, as indicated by the magenta curve in Fig. 9.1, and thus to make more systems, in particular enterprises, amenable to analytic study.

9.2.1.2 The Ontological System Concept

As a first step in reducing the complexity of systems, we make a distinction between homogeneous and heterogeneous systems. Every non-trivial system is a heterogeneous system, which means that it is some, possibly complicated, combination of homogeneous systems. For example, a human being is a physical system but also a chemical and a biological one, and as a whole it is also a social individual. In order to address its complexity, Bunge considers a *heterogeneous* system as a layered nesting of homogeneous systems [4]; he suggests studying the composing homogeneous systems first, leaving the study of the complex heterogeneous totality for later. Hereafter, whenever the term "system" is used, a homogeneous system is meant, according to the following definition [4]:

A (homogeneous) *system* can be conceived as a triple $(\mathbb{C}, \mathbb{E}, \mathbb{S})$, where:

$\mathbb{C}$ is a set of elements, all belonging to the same category,

called the *composition* of the system;

$\mathbb{E}$ is a set of elements of the same category as the elements in $\mathbb{C}$,

called the *environment* of the system;

$\mathbb{S}$ is a set of influencing bonds among the elements in $\mathbb{C}$ and between them and the elements in $\mathbb{E}$,

called the *structure* of the system.

Figure 9.2 depicts this definition. The red- and purple-coloured boxes are elements in $\mathbb{C}$, and the green ones are elements in $\mathbb{E}$. The purple-coloured closed curve depicts the *boundary* of the system. It is defined as the subset of $\mathbb{C}$ for whose elements it holds that they are connected by structural links with elements in $\mathbb{E}$, in correspondence with [6]. The elements in $\mathbb{C}$ that are not connected to elements in $\mathbb{E}$ are called *kernel* elements. Hereafter, we will call the triple $(\mathbb{C}, \mathbb{E}, \mathbb{S})$ the *construction* of a system.

There are three important comments to be made about the system definition above. The first one is that a structural link between two elements means that one of them acts upon the other, or that both do, as discussed in [7]. The second one is that every element in $\mathbb{C}$ must act upon or be acted upon by at least one other element in $\mathbb{C}$, so that all elements in $\mathbb{C}$ are directly or indirectly connected. Consequently, isolated elements, or isolated clusters of connected elements, cannot exist; their presence would violate the basic notion of system. The third comment is that the elements in $\mathbb{C}$ and $\mathbb{E}$ are of the same category. Examples of system categories are: physical, chemical, biological, and social. Only systems of the same category can interact, systems of different categories cannot. For example, if you have something in your mind that you want to 'tell' your computer in order to not forget it, it is ultimately your homogeneous physical system (which is a part of your heterogeneous entirety) that interacts with the homogeneous physical system of the heterogeneous computer system, in particular through the physical forces that your fingers

exert on the keys of the keyboard. The pressing of a key causes the generation of a train of electrical signals that carry the code of the key that is pressed. This sequence of signals is transmitted to etc. etc.

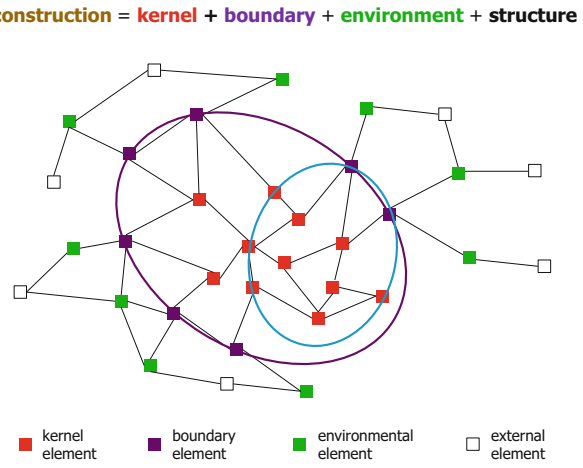

Fig. 9.2 Depiction of the construction of a system and of a subsystem

Based on the provided definition of system, the next definition of subsystem is in force [4]: A thing x is a *subsystem* of a system y if and only if x is a system, and if:

$$\mathbb{C}(x) \subseteq \mathbb{C}(y)$$
$$\mathbb{E}(x) \subseteq (\mathbb{C}(y) \setminus (\mathbb{C}(x)) \cup \mathbb{E}(y)$$
$$\mathbb{S}(x) \subseteq \mathbb{S}(y)$$

The blue-coloured closed curve in Fig. 9.2 depicts the boundary of a subsystem of the system whose boundary is depicted by the purple closed curve. As a corollary, every system may have many subsystems, and can be subsystem of many systems. Note that one cannot just view something as a system. Only systems, according to the definition above, can be 'viewed' as a system. All other things can't.

Both its elements and its subsystems are called *components* of a system. Consequently, the composition of a system comprises both its elements, so the members of $\mathbb{C}$, and all subsystems that one likes to distinguish. Therefore, the composition of a system may be said to consist of *elementary* components and *composite* components, keeping in mind that the latter are always built up of elementary components. As a corollary, the structural bonds between two composite components are actually structural bonds between elements in one composite component and elements in the other.

Likewise, the environment of a system comprises both its elements, so the members of $\mathbb{E}$, and all subsystems, built up of these elements, that one considers useful to distinguish. In addition, the structural bonds between composite components in $\mathbb{C}$ and $\mathbb{E}$ are actually structural bonds between elements in $\mathbb{C}$ and elements in $\mathbb{E}$.

9.2.2 Conceptual Models of Concrete Systems

9.2.2.1 Introduction

According to the MU theory (cf. Chap. 6), a conceptual model of a concrete system is a conceptualisation of the system within an appropriate conceptual schema or meta model. The conceptual model is said to be an instance of the conceptual schema. Hereafter, we use the word "model" to refer to a model (instance) as well as to a meta model, following the current (confusing) practice in conceptual modelling.

We will distinguish three sorts of conceptual models: black-box models, grey-box models, and white-box models. They will be discussed in Sects. 9.2.2.2, 9.2.2.3, and 9.2.2.4, respectively. Their distinction is related to the fundamental difference between the function and the construction perspective on things, as explained in the TAO theory (cf. Chap. 7). Figure 9.3 shows the construction perspective on a car (right), and a particular function perspective, namely the driving function (left). Both models are extensively discussed in Chap. 7.

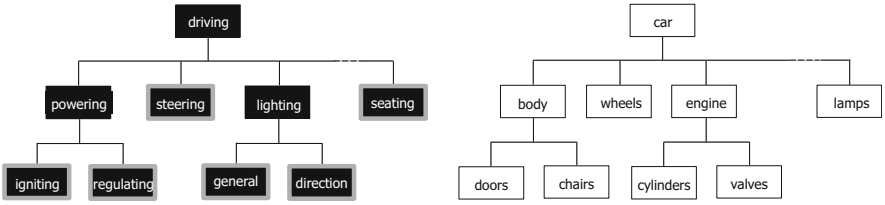

Fig. 9.3 The function and the construction perspective on cars

White-box models are suited for studying the construction and the operation of systems. The white-box model on the right side of Fig. 9.3 is actually only the decomposition of a car in its subsystems, sub-subsystems, etc., disregarding the structure of the system. It resembles a Bill-of-Material (BoM). *Black-box models* are suited for studying the behaviour of systems and their possible function(s). The black-box model on the left side of Fig. 9.3 is the decomposition of the driving function of a car into subfunctions, sub-subfunctions, etc. *Grey-box models* are black-box models with an internal state. Examples will be given in Sect. 9.2.2.3.

With reference to Fig. 9.1, the only thing one can do in the region of unorganised complexity is black-box modelling, and the best one can hope for is to find correlations between (functional) variables. In the region of organised simplicity, one has the option to apply white-box modelling, and thus to discover the causal relationships between system acts and observable effects. Consequently, one can acquire a deeper understanding of a system, not only from the construction perspective, but also from the function perspective.

9.2.2.2 The Black-Box Model

A *black-box* model of a concrete system is a conceptual model of the system that disregards completely the construction and the operation of the system (this explains the name "black box"). Therefore, black-box models are only suited to study the behaviour of systems, expressed in relationships between the functional variables that one chooses to study. Well-known examples of black-box models are economic models. They are commonly expressed in differential equations concerning a number of (economic) variables.

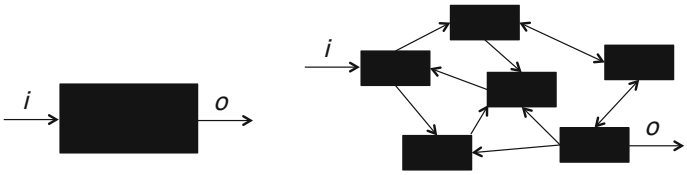

Fig. 9.4 The black-box model (left) and a decomposition (right)

Basically, the knowledge of a concrete system that is contained in a black-box model is the relationship between the input flow i and the output flow o in the course of time (t). In formal notation: $o = B(i, t)$. Both flows are time series of values of (functional) variables. Figure 9.4 (left side) exhibits the common graphical representation of a black-box system. The *behaviour function* B is often not or only partially known, meaning that one only knows that the output flow o at time t is somehow the effect of the input flow i before t. It is always possible to decompose a black-box model into a network of connected black-box models (cf. Chap. 7). A well-known example of such a functional decomposition is the *control model* [8], as applied for example in cybernetics and biology.

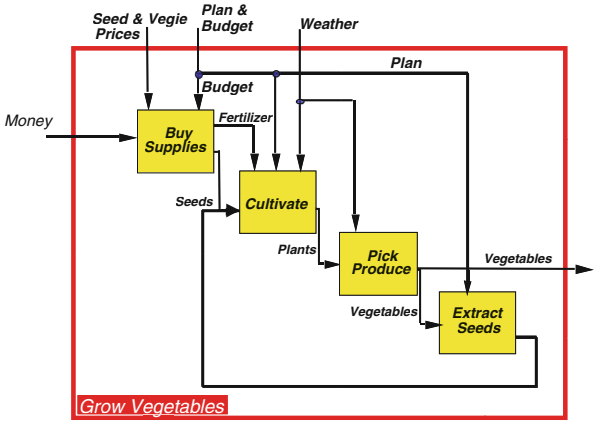

Fig. 9.5 Example of an SADT activity diagram

A widely used technique for representing black-box models, including their decomposition, is SADT (Structured Analysis and Design Technique), developed by Douglas T. Ross [9], and included in many structured analysis and design methods. Figure 9.5 shows an example of an SADT activity diagram. It is important to recognise that the diagram represents only a functional understanding of the activity of growing vegetables. There is no hint whatsoever to the construction of a system that is able to exhibit this behaviour. As another example, it is not very difficult to make a functional model of a coffee machine along the lines of Fig. 9.5. In fact, we all have some functional (black-box) model of the systems we daily use, like coffee machines, ATMs, and cars. As is illustrated for cars in Sect. 9.2.2.1 and in Chap. 7, there is no straightforward mapping between functional (black-box) models and constructional (white-box) models, because they are of a fundamentally different nature.

9.2.2.3 The Grey-Box Model

The *grey-box* model is a black-box model with an internal state [4]. Consequently, the behaviour is now determined by three variables: the flow of input items i, the flow of output items o, and the state of the system s, next to time (t). The behaviour function is formally defined as: $o = B(i, s, t)$. Figure 9.6 exhibits the common graphical representation of a grey-box model (left side) and of a possible decomposition (right side).

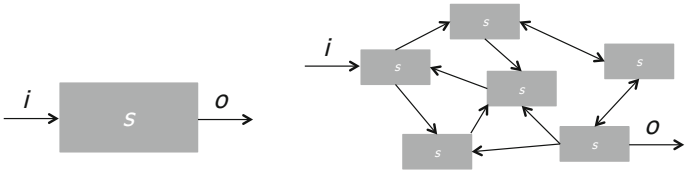

Fig. 9.6 The grey-box model (left) and a decomposition (right)

A well-known specialisation of the grey-box model is the finite automaton (FA), often also called finite state machine (FSM). A *finite automaton* is a mathematical model of a system with discrete inputs and outputs, and with discrete states. By this is meant that the system is at any point in time in some state, and that the state space is finite (or denumerable infinite). State changes or transitions occur on an input from a finite (and commonly small) set of possible inputs. Usually, there is an initial state and there are one or more final or terminal states. FAs are mostly associated with the way in which they are commonly represented, namely the *state transition diagram* (STD), which we will use in Fig. 9.8.

Another well-known specialisation of the grey-box model is the discrete event system (DES). A *discrete event system* is a discrete-state, event-driven conceptual system (cf. Chap. 6). The notion of *discrete-state* is similar to the notion of state in an

FA. By *event-driven* is meant that the system responds to the occurrence of particular input events [10]. The number of distinct events may be denumerable infinite. Many concrete systems, in particular logistic systems, information systems, and organisations, can be conceived as discrete event systems. The notion of discrete event system is often considered to be identical to the notion of grey-box model [4]. This is not true, however, because in a grey-box model one does not require that the input to which the system responds, consists of discrete events. As an example, the variables in economic (grey-box) models are mostly not discrete but continuous.

Typical problems that are studied by means of discrete event models are stochastic systems, like customers in supermarkets and cars at fuel stations. To get deeper insight into the behaviour function, discrete event simulation is often applied [10].

The grey-box model of a discrete event system can mathematically be defined as a tuple (**B**, **O**, **I**, **S**), where:

B is a partial function, called the *behaviour function*
O is a set of items, called the *output base*
I is a set of items, called the *input base*
S is a set of items, called the *state base*
B is defined as: $\wp\mathbf{I} * \wp\mathbf{S} \rightarrow \wp(\mathbf{O} * \mathbb{D})$

The (mathematical) extension of **B** is a set of rules of the form (*I*, *S*, *R)*, where:

I is the current input; $I \subseteq \mathbf{I}$
S is the current state; $S \subseteq \mathbf{S}$
R is the response: a set of pairs (o, d) with $o \in \mathbf{O}$ and $d \in \mathbb{D}$; d is a time delay; its effect is that o becomes existent at the point in time $t =$ Now + d, where Now is the time of executing the rule.

(Note: for an explanation of the mathematical symbols, see Sect. 9.2.3.1.)

Illustration: Traffic Control System

To illustrate the grey-box system, more specifically the FA, we take a traffic control system (TCS) at a simple crossover, as shown on the left side in Fig. 9.7. Suppose that you are asked to produce first a black-box model and then a grey-box model of the TCS. The black-box model that you may arrive at, after having observed the traffic control system for some time, is shown to the right of it. Theoretically, there are nine possible outputs: R1R2, R1G2, R1Y2, G1R2, ~~G1G2~~, ~~G1Y2~~, Y1R2, ~~Y1G2~~, and ~~Y1Y2~~, where R denotes red light, G denotes green light, Y denotes yellow light, 1 denotes Cycle 1, and 2 denotes Cycle 2. Four of them do not occur however, and are therefore struck out above. The five remaining outputs are shown on the right side in Fig. 9.7. The arrows indicate the order in which the outputs occur. The additional knowledge of the behaviour function B, which you can deduce from observing the traffic lights and the traffic, is that the transition R1G2 → R1Y2 is influenced by arriving and/or waiting traffic in Cycle 1 (but you don't know exactly how). Likewise, the transition G1R2 → Y1R2 is influenced by arriving and/or waiting traffic in Cycle 2.

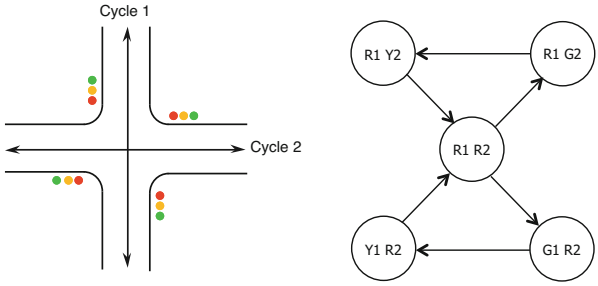

Fig. 9.7 Picture of the TCS (left) and its black-box model (right)

In order to produce a grey-box model, one has to conceive an internal state of the system. With this new meta model in mind, one is able to acquire the next, advanced, knowledge (cf. Fig. 9.8). If the output R1G2 is produced, it will stay for at least a minimum amount of time. Let us call this the (standard) *move time* for Cycle 2, abbreviated to MT2. As long as there is no traffic waiting in Cycle 1, this output is prolonged. However, as soon as the (standard) move time for Cycle 2 has passed, and there is traffic waiting for red light in Cycle 1, the output R1Y2 will be produced. This output appears to hold on for a fixed amount of time, which we will call the *stop time* for Cycle 2 (ST2). After the stop time has elapsed, the light in Cycle 2 becomes red. However, also the light in Cycle 1 remains red for some fixed amount of time. Let us call this amount of time the *clear time* of Cycle 2 (CT2), meaning that it is meant for clearing the crossing from traffic in Cycle 2. The output R1R2 is produced. After the clear time has elapsed, the output G1R2 is produced. And then the whole story is repeated, with the cycles exchanged. In order to cope with these observations, you distinguish the next different states: W1M2, W1P2, W1S2, C1W2, M1W2, P1W2, S1W2, and W1C2, where W stands for "waiting", M for "(standard) moving", P for "prolonged (moving)", S for "stopping", and C for "clearing". The state transitions occur in the order as exhibited in Fig. 9.8. At the top of the figure, the outputs (traffic lights) are shown that correspond with the states of the system.

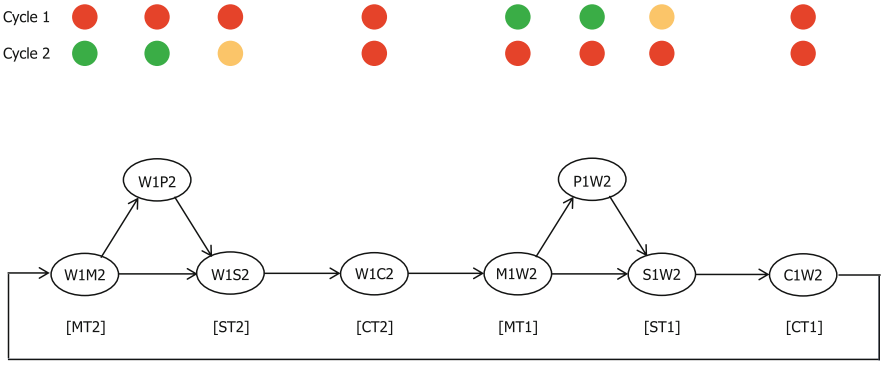

Fig. 9.8 STD of the grey-box model of the TCS

The choice between transiting from W1M2 to W1P2 or to W1S2 depends on the presence of traffic in Cycle 1. If traffic arrives in the state W1M2, then the transition to W1S2 is made, but only after MT2 time units have elapsed since the beginning of the state W1M2. Otherwise the transition to W1P2 is made. As soon as traffic in Cycle 1 arrives in this state, the transition to W1S2 is made immediately. The system remains in this state for ST2 time units. Then, the transition to C1W2 is made, which takes CT2 time units. After this time has elapsed, the transition to M1W2 is made. Similar observations hold for the transitions from M1W2 to P1W2 or S1W2.

9.2.2.4 The White-Box Model

The white-box model of a system is a conceptual model that allows one to study the *construction* of the system, thus the triple ($\mathbb{C}$, $\mathbb{E}$, $\mathbb{S}$) as discussed in Sect. 9.2.1.2, as well as its *operation*, that is, the way in which the elements in $\mathbb{C}$ and the elements in $\mathbb{E}$ interact, through the bonds in $\mathbb{S}$. Consequently, one is able to reveal the 'mechanism' that makes it 'tick'. The behaviour that the 'mechanism' causes to occur can be studied with a grey-box (or black-box) model of the system, as we have seen above.

For a proper discussion of the white-box system, we introduce the notion of world. With every white-box model of a concrete system, a *world* is associated, where the acts of the (elements of the) system have their effect. At any point in time, the world of a system is in some state. A *state* is simply defined as a set of facts. A state change is called a *transition*. It consists of the addition and/or removal of one or more facts.

The state of a white-box system differs from the state of a corresponding grey-box system. For a grey-box system, the state concept is an instrument to better understand the behaviour of the system. For a white-box system, the state concept is an instrument to better understand its operation. The relationship between behaviour and operation is that the behaviour of a concrete system (which can be studied by using a black-box or grey-box model) is brought about by the operation of its construction (which can only be studied by using a white-box system).

Next to the state of the world of a system, there is the state of the system itself. By the *state of a system* at a point in time t we understand the particular triple ($\mathbb{C}$, $\mathbb{E}$, $\mathbb{S}$) at time t (cf. Sect. 9.2.1.2). In the course of time, the composition or the environment or the structure may change. Within EE, such changes are considered to be the effect of acts by another system, whose world contains facts that represent the elements in $\mathbb{C}$, $\mathbb{E}$, and $\mathbb{S}$. This other system is commonly called the governance system of the system under consideration [11, 12]. In the DELTA theory, we confine ourselves to studying systems in some system state, that is, we assume a fixed construction ($\mathbb{C}$, $\mathbb{E}$, $\mathbb{S}$). Proper illustrations of the white-box model will be provided after the PRISMA model is discussed (in the next section).

9.2.3 The PRISMA Model

9.2.3.1 Introduction

In the next subsections, we will present and discuss a meta model for discrete event systems called the PRISMA model, which builds on the SMART model [13]. It comprises a grey-box and a white-box (meta) model. The PRISMA model is suited for studying discrete event systems, both *technical systems*, thus systems in which the elements are non-human, and *social systems*, that is, systems in which the elements are human. An important subclass of social systems is organisations, as discussed in the PSI theory (cf. Chap. 8). Therefore, one may consider the PRISMA model as a (mathematical-logical) formalisation of the PSI theory.

In addition, it appears that many technical systems are actually social systems, only technically implemented. Well-known examples are (automated) enterprise information systems (cf. Chap. 11), but also ATMs, elevator control systems, vending machines, web shops, and traffic control systems. Consequently, these systems can be very well understood and studied within the PRISMA model.

We distinguish three ways in which systems influence each other, called activating, restricting, and impeding. By *activating* is understood that a system creates events to which other systems respond by creating state changes and/or events. When responding to an event, a system takes the current state of its world into account. The state of a system's world can be changed by the system itself, but also by other systems. This passive way of mutual influencing is called *restricting*, since the effect is that a system's response space is restricted (note that a system is not 'aware' of state changes until it is activated—only then will it 'see' the new state). By *impeding* is understood that a system creates events for whose occurrences other systems have to wait before they can continue what they were doing. Because of the three ways of mutual influencing, systems within the PRISMA model are said to communicate asynchronously.

In the remainder of this document, set theory and logic are applied when considered useful. For the convenience of the reader, we list below the symbols that are used, with their meanings.

x	in general, small letters denote elements (of sets)
X	in general, capital letters denote sets
$\in$	membership of a set; $x \in A$ means that x is an element of A
$\notin$	negation of membership; $x \notin A$ means that x is not an element of A
$\varnothing$	empty set; $A = \varnothing$ means that for all x: $x \notin A$
$\subseteq$	subset; $A \subseteq B$ means that A is a subset of B; for all x: $x \in A \Rightarrow x \in B$
$\cup$	union; $A \cup B$ is the set of elements x for which holds: $x \in A$ or $x \in B$
$\cap$	intersection; $A \cap B$ is the set of x for which holds: $x \in A$ and $x \in B$
\	set difference; A\B is the set of elements x for which holds: $x \in A$ and $x \notin B$
Δ	symmetric set difference; $A \Delta B = (A \setminus B) \cup (B \setminus A)$
$*$	Cartesian product of a set; $A*B$ is the set of tuples (x, y) with $x \in A$ and $y \in B$

$\wp$	powerset; $\wp$A is the set of all subsets of A
f: A → B	(mathematical) function f with domain A and range B.
x	variable to denote a type (of acts or facts); a type is a unary predicate in logic, for example, person or dog.
X	variable to denote the class that is the extension[1] of the type **x**
$\underline{\mathbf{X}}$	variable to denote the union of the extensions of all $\mathbf{x} \in \mathbf{X}$
$\wedge$	logical conjunction (also denoted by **and**)
$\vee$	logical (inclusive) disjunction (also denoted by **or**)
$\underline{\text{ct}}(f)$	creation time of fact f
$\underline{\text{et}}(f)$	effectuation or event time of fact f

For a proper discussion of the PRISMA model, a discrete linear time dimension is adopted, which means that we consider the time axis to be divided into distinct *time units* of arbitrary but equal length.[2] Every such time unit on the time axis is called a *point in time*. Events only occur on (or in) these points in time, and they take place instantaneously, that is, within the duration of the point in time. An *event* is defined as the becoming existent (or ceasing to exist) of a fact at some point in time. As mentioned above, facts have a *creation time* (ct) and an effectuation or *event time* (et).

The notion of a *discrete linear time scale*, for any time unit, can be formalised in the following way:

$\mathbb{R}$	the (ordered) set of real numbers
$\mathbb{N}$	the (ordered) set of natural numbers
$\mathbb{T} : \mathbb{N} \rightarrow \mathbb{R}$	the (ordered) set of discrete points in time; we will use t_n as a shorthand for $\mathbb{T}(n)$; the time difference between any t_{n+1} and t_n is the same; it is called the time unit (*tu*)
Now	the current point in time; Now $\in \mathbb{T}$, so Now is always some t_n
$\mathbb{D}$	the set of (positive) time durations; for every $d \in \mathbb{D}$, it holds that $d = k * tu$ with $k \in \mathbb{N}$

From now on, we mean by a point in time t an element $t_n \in \mathbb{T}$.

9.2.3.2 The PRISMA Grey-Box Model

The distinction that we have made in the PSI theory (cf. Chap. 8) between the things that constitute its core, thus its production (or P-) acts/facts, and the things that serve to make them happen, thus the coordination (or C-) acts/facts, appear to have a general applicability. Therefore, the same distinction is made in the PRISMA model. The only difference is that they are now not connected in larger structures of conversations or transactions.

[1] The extension of a type is the set of objects that conform to the type (cf. Chap. 5).

[2] The duration or length of the applied time unit will depend on the application domain. Therefore, it may vary from nanoseconds or microseconds (for technical systems) to hours or days (for enterprises).

The PRISMA grey-box model can best be considered as an extended 'normal' grey-box model, as discussed in Sect. 9.2.2.3. The extension is that it comprises *process dependency* next to *state dependency*. It means that the response to an input item does not only depend on the current state, but may also depend on the occurrences of (past or future) input or output events.

A *prisma* can formally be defined as a tuple (**P**, **R**, **I**, **S**, **M**, **A**), where:

P is a partial function, called the *performance function*
R is a set of C-fact types, called the *reaction base*
I is a set of C-fact types, called the *impediment base*
S is a set of P-fact types, called the *state base*
M is a set of P-fact types, called the *mutation base*
A is a set of C-fact types, called the *action base*

P is defined as: $\wp((\underline{\mathbf{A}} \cup \underline{\mathbf{I}}) * \mathbb{T}) * \wp(\underline{\mathbf{S}} * \mathbb{T}) \rightarrow \wp(\underline{\mathbf{M}} * \mathbb{D}) * \wp(\underline{\mathbf{R}} * \mathbb{D})$

At every point in time, a prisma disposes of a set of *agenda*.[3] An agendum is a pair (c, t), in which c is a C-fact and t is its event time. The C-fact belongs to the extension[4] of the *action base* or the *impediment base*. The set of agenda c with $\underline{\text{et}}(c) = t$, is called the *trigger* at time t (note: commonly the trigger will be a singleton set). Considering impediments as potential agenda allows us to deal with impediments in a simple way: we consider them to be agenda, similar to real agenda, that is, to elements in the extension of the action base. If the settling of a real agendum, that is, a pair (c, t), in which c is an element of $\underline{\mathbf{A}}$, has to wait for an impediment, the prisma will skip this agendum and settle the impediment, once it has occurred. The having occurred of the real agendum then becomes a state condition for settling the impediment event.

A prisma responds to a trigger instantaneously, so within the duration of the point in time t. As the effect of settling a trigger, a finite set of P-events is created, called the *mutation*, and a finite set of C-events, called the *reaction*. The set of P-fact types, whose instances can belong to a mutation, is called its *mutation base*. The set of C-fact types, whose instances can belong to a reaction, is called the *reaction base*.

The response to a trigger is generally dependent on the state of the P-world, which is a set of P-facts. The set of P-fact types, instances of which can belong to a state, is called the *state base* of the prisma.

The (mathematical) extension of **P** is a set of *performance rules* of the form (A, S, M, R) where:

A is the *agenda*; $A \subseteq (\underline{\mathbf{A}} \cup \underline{\mathbf{I}}) * \mathbb{T}$
the *trigger* at time t is $\{f \in A: \underline{\text{et}}(f) = t\}$
S is the *state*; $S = \{(f, t)$ with $f \in \underline{\mathbf{S}}$ and $t \in \mathbb{T}\}$
M is the *mutation*: a set of pairs (m, d) with $m \in \underline{\mathbf{M}}$ and $d \in \mathbb{D}$
R is the *reaction*: a set of pairs (r, d) with $r \in \underline{\mathbf{R}}$ and $d \in \mathbb{D}$

[3] The word 'agenda' is the plural form of the Latin word 'agendum', meaning 'thing to be done'. So, agenda are 'to do' items.

[4] The extension of a type is the set of objects that conform to the type (cf. Chap. 5).

The delay d in a pair (m, d) or (r, d) is an occurrence delay; it means that the P-fact m or the C-fact r becomes effective at the event time $t = \text{Now} + d$, where Now is the point in time at which the performance rule is executed. As said, $\{f \in A: \underline{\text{et}}(f) = t\}$ is commonly a singleton set, thus a set containing only one element. Note that the state S at time t comprises existing facts (with $\underline{\text{et}}(f) \leq t$) and future facts (with $\underline{\text{et}}(f) > t$).

On the basis of their formal definition, the mutual influencing of prismas as discussed in Sect. 9.2.3.1, can be described more precisely as follows.

A prisma1 *activates* a prisma2 if $\underline{\mathbf{R}}_1 \cap \underline{\mathbf{A}}_2 \neq \emptyset$ (so if $\underline{\mathbf{R}}_1$ and $\underline{\mathbf{A}}_2$ overlap). If this is the case, then all C-events in a reaction of prisma1, of which the C-fact belongs to this intersection, are instantly added to the agenda of prisma2. If prisma1 and prisma2 are identical, we speak of *self-activation*. Through self-activation, periodic activities can be modelled conveniently. If this is the case, the period equals the settlement delay.

A prisma1 *restricts* a prisma2 if $\underline{\mathbf{M}}_1 \cap \underline{\mathbf{S}}_2 \neq \emptyset$ (so if $\underline{\mathbf{M}}_1$ and $\underline{\mathbf{S}}_2$ overlap). If this is the case, then every P-event in a mutation of prisma1 of which the P-fact belongs to this intersection, affects the state of prisma2, at its event time. The way in which the state of prisma2 is affected by a mutation is defined as follows. If *S1* is the state of prisma2 before applying a mutation *M* (at its event time), and *S2* is the state afterwards, then $S2 = S1 \,\Delta\, M$. (Δ is the symmetric set difference, cf. Sect. 9.2.3.1). If prisma1 and prisma2 are identical, we speak of *self-restricting*. This is the classical concept of the state of a world (where only the system itself can make changes).

A prisma1 *impedes* a prisma2 if $\underline{\mathbf{R}}_1 \cap \underline{\mathbf{I}}_2 \neq \emptyset$ (so if $\underline{\mathbf{R}}_1$ and $\underline{\mathbf{I}}_2$ overlap). If this is the case, then all C-events in a reaction of prisma1 of which the C-fact belongs to this intersection, are instantly added to the impediments of prisma2. Consequently, prisma2 may have to wait with responding to an action until one or more impediments have occurred. Self-impeding is ignored, because it doesn't seem to make sense.

If the action base **A** of a prisma consists of one fact type, the prisma is called *elementary*. The action bases of elementary prismas are disjoint. A *composite* prisma is a collection of elementary prismas. The specification of a composite prisma in terms of its constituting elementary prismas is simple: every component of the tuple (**P**, **R**, **I**, **S**, **M**, **A**) of a composite prisma is equal to the set-theoretic union of the corresponding components of the constituting elementary prismas.

Illustration: Traffic Control System

Let us take the Traffic Control System from Sect. 9.2.2.3 to exemplify the PRISMA grey-box model. From the STD in Fig. 9.8 and the accompanying explanation, we deduce the next components of the tuple (**P**, **R**, **I**, **S**, **M**, **A**):

A = {let_pass(Cycle)}
I = ∅
S = {phase(Cycle), move_time(Cycle), stop_time(Cycle), clear_time(Cycle)}
R = ∅
M = {phase(Cycle)}

In this specification, the variable Cycle has the value cycle1 or cycle2. The value of phase(Cycle) is W1M2 or W1P2 or W1S2, etc. let_pass(Cycle) is the external trigger to which the prisma responds (cf. Fig. 9.8). A phase change takes place if the mutation of the prisma contains a new P-fact of the type phase(Cycle). Next to the

current phase of each of the cycles, the state also includes the current values of the parameters (move_time, stop_time, and clear_time) for each of the cycles.

Table 9.1 exhibits the performance function **P** that can be deduced from the grey-box model in Sect. 9.2.2.3. The *agenda* column contains the triggers to settle. The *state* column contains logical propositions concerning the state of the production world, and the *mutation* column contains the state changes to be effectuated.

The table presents the situation that traffic is arriving in cycle1. A similar table applies for the situation that traffic is arriving in cycle2, by exchanging cycle1 and cycle2. Occurrence delays are specified by a value between "[" and "]". If no occurrence delay is specified, the default value is assumed (which is 1 time unit). The abbreviations have the following meanings: MT (Cycle) stands for the standard move time in Cycle, ST (Cycle) for the stop time in Cycle, and CT (Cycle) for the clear time in Cycle. The meaning of the delay D is explained later on.

Table 9.1 The performance function of the TCS

agenda	state	mutation
let_pass(cycle1)	phase(cycle1) = waiting phase(cycle2) = moving **or** prolonged_moving) **and there is no future event** phase(cycle2) = stopping)	phase(cycle2) := stopping [D] phase(cycle2) := waiting [D + ST(cycle2)] phase(cycle1) := moving [D + CT(cycle2)]

The trigger to be settled is let_pass(cycle1). If the current state (at time t) comprises the facts <phase(cycle1) = waiting> and <phase(cycle2) = moving or prolonged_moving>, both with an event time in the past or present ($\underline{et}(f) \leq t$), and there is not already a future stopping for cycle2 (with $\underline{et}(f) > t$, caused by another car), then the rule is executed; otherwise nothing happens. The response of executing the rule is the specified mutation. It says that the phase of cycle2 will become 'stopping' after D time units. The delay D is defined as follows: D = max (0, (MT (cycle2) − (Now − ETM)), where ETM = $\underline{et}$(phase(cycle2) = moving), the point in time at which the phase of cycle2 started to be moving. The mathematical expression is clarified in Fig. 9.9. If the current time is Now1, then the delay is the time represented by the blue line. If the current time is Now2, then the delay is zero.

In addition, the mutation contains the state changes phase(cycle2) := waiting [D + ST(cycle2)] and phase(cycle1) := moving [D + CT(cycle2)]. These occurrence delays are also clarified by Fig. 9.9.

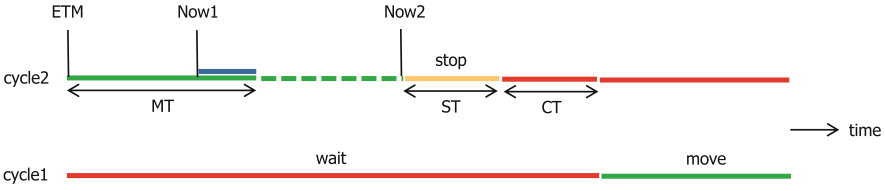

Fig. 9.9 Explanation of the time delay D

9.2.3.3 The PRISMA White-Box Model

The influencing relationships among a collection of prismas can be made more comprehensible if the collection is conceived as a prismanet. A *prismanet* is a white-box system, wherein the construction $(\mathbb{C}, \mathbb{E}, \mathbb{S})$ comprises three kinds of components: processors, banks, and channels. The components in the composition $\mathbb{C}$ and in the environment $\mathbb{E}$ are processors. The structure $\mathbb{S}$ consists of banks and channels and of the various links that connect them with the processors in $\mathbb{C}$ and in $\mathbb{E}$.

Processors are the 'motors' of prismas. The motor of an elementary prisma is an *elementary processor*, and the motor of a composite prisma is a *composite processor*. The operation of a processor is fully determined by the performance function **P** of the corresponding prisma (i.e. the prisma of which it is the motor).

Channels are used to transmit and keep C-events. A channel Cn is determined by its *transmission base* TB, which is the set of C-fact types whose instances it can keep and transmit. The set of C-events in a channel at time t, is called the *contents* of the channel at time t. The channel metaphor runs as follows. Suppose processor Pi creates at time t the pair (c, d), where c is a C-fact and d is the occurrence delay. The metaphor then is that Pi 'puts' c in a channel at time $\underline{ct}(c)$ (the creation time of c) and that c 'arrives' at a processor P_j at the event time $\underline{et}(c) = t + d$. On arrival, it is settled instantaneously. Every transmitted C-event remains in the channel, because it may be an impediment for one or more (other) prismas. If the transmission base consists of one C-fact type, the channel is called a *single channel*. The transmission bases of the single channels in a prismanet are disjoint. A collection of single channels is called a *multiple* channel. The transmission base of a multiple channel is the union of the transmission bases of the composing single channels.

Banks are used to keep P-events. A bank Bk is determined by its *contents base* CB, which is the set of P-fact types whose instances it can contain. The set of P-events in the bank at some time is called the *contents* of the bank at that time. If the contents base consists of one P-fact type, the bank is called a *single bank*. The contents bases of the single banks in a prismanet are disjoint. A collection of single banks is called a *multiple* bank. The contents base of a multiple bank is the union of the contents bases of the composing single banks.

A channel C_n is called an *action channel* of processor P_j if the transmission base of the channel is a subset of the action base of the prisma of which processor P_j is the motor, so if $TB(C_n) \subseteq \mathbf{A}_j$. The settling of an action may be impeded by one or more C-events or P-events. It means that the processor has to wait until these events have occurred. A Processor P_j is impeded by C-events in a channel C_n if the transmission base of the channel is a subset of the impediment base of the prisma of which processor P_j is the motor, so if $TB(C_n) \subseteq \mathbf{I}_j$. If so, the channel C_n is called an *impediment channel* of processor P_j.

As the result of settling an action, processor P_i creates a (possibly empty) set of P-events, called the mutation. They are put in every bank B_k of which the contents base is a subset of the mutation base of the prisma whose motor is processor P_i, so of which $CB(B_k) \subseteq \mathbf{M}_i$. If so, bank B_k is called a *mutation bank* of processor P_i.

In addition, processor P_i creates a (possibly empty) set of C-events, called the reaction. They are put in every channel C_n of which the transmission base is a subset of the response base of the prisma whose motor is processor P_i, so of which TB (C_n) $\subseteq$ $\mathbf{R}_i$. Consequently, channel C_n is called a *reaction channel* of processor P_i.

When dealing with a C-event, a processor P_j may take P-events that are kept in one or more banks into account. This holds for bank B_k if its contents base is a subset of the state base of the prisma of which processor P_j is the motor, so if CB (B_k) $\subseteq$ $\mathbf{S}_j$. Consequently, bank B_k is called an *inspection bank* of processor P_j.

The *operation* of a processor must be understood as follows. Processors constantly loop through their *operating cycle*, of which the cycle time is equal to or less than the time unit (cf. Sect. 9.2.3.1). In every cycle, the processor 'sees' the current trigger (if any) and brings about a response by evaluating the corresponding performance rule.

9.2.3.4 The Prismanet Diagram

The understanding of a prismanet may be enhanced by expressing it in a *prismanet diagram* (cf. Fig. 9.10).

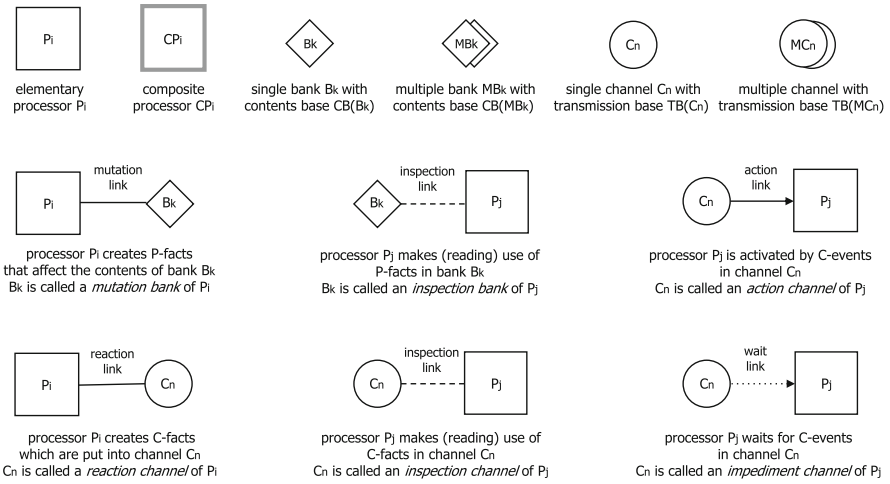

Fig. 9.10 Legend of the prismanet diagram

Processors, banks, and channels are respectively represented by boxes, diamonds, and disks, as shown in the top part of the figure.

Channels are connected to processors by four kinds of links: reaction links, action links, inspection links, and wait links. A *reaction link* connects a processor with one of its reaction channels. An *action link* connects a processor with one of its action

channels. An *inspection link* connects a processor with one of its inspection channels. A *wait link* connects a processor with one of its impediment channels. Banks are connected to processors by two kinds of links: mutation links and inspection links. A *mutation link* connects a processor with one of its mutation banks. An *inspection link* connects a processor with one of its inspection banks.

From a prismanet diagram, one can directly deduce that for an elementary prisma with processor P_j as its motor:

- The action base $\mathbf{A}_j$ is the union of the transmission bases of its action channels.
- The impediment base $\mathbf{I}_j$ is the union of the transmission bases of its impediment channels.
- The state base $\mathbf{S}_j$ is the union of the content bases of its inspection banks and the transmission bases of its inspection channels.
- The reaction base $\mathbf{R}_j$ is the union of the transmission bases of its reaction channels.
- The mutation base $\mathbf{M}_j$ is the union of the content bases of its mutation banks.

Illustration: Traffic Control System

To illustrate the PRISMA white-box model for the traffic control system (TCS), Fig. 9.11 exhibits its prismanet diagram. The light-grey coloured frame represents the Scope of Interest (SoI). It means that one is exclusively interested in the operation of the processors within the SoI. They are therefore called internal processors, whereas the composite processors CP_1 (traffic participant) and CP_2 (traffic control manager) are called environmental processors. To show this, their boxes are coloured light-grey. Channel C_1 is a reaction channel of CP_1 and an action channel of P_1. Bank B_1 is a mutation bank and an inspection bank of P_1, as well as an inspection bank of CP_1. The banks B_2, B_3, and B_4 are mutation banks of CP_2 and inspection banks of P_1.

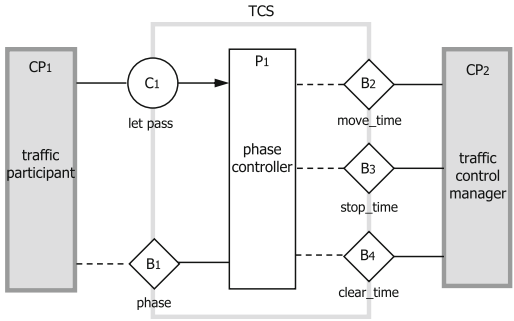

Fig. 9.11 Prismanet diagram of the TCS

The traffic participants take note of the phase of each cycle (by looking at the traffic lights) and generate let_pass commands (by passing a sensor in the road). There is a traffic control manager, who is able to change the control parameters of each of the cycles: move time, stop time, and clear time (through updates of their values).

Processor P_1 responds to let_pass commands by bringing about the appropriate phase changes, according to the performance function that is exhibited in Table 9.1.

From the diagram in Fig. 9.11, together with the discussion in Sect. 9.2.3.2 and the explanation in Fig. 9.9, one can easily verify the following specifications of the action base, the impediment base, the state base, the mutation base, and the reaction base of the prisma with motor P_1:

$\mathbf{A}_1 = \mathrm{TB}(C_1) = \{\text{let\_pass(Cycle)}\}$
$\mathbf{I}_1 = \varnothing$
$\mathbf{S}_1 = \mathrm{CB}(B_1) \cup \mathrm{CB}(B_2) \cup \mathrm{CB}(B_3) \cup \mathrm{CB}(B_4) = \{\text{phase(Cycle), move\_time(Cycle), stop\_time(Cycle), clear\_time(Cycle)}\}$
$\mathbf{M}_1 = \mathrm{CB}(B_1) = \{\text{phase(Cycle)}\}$
$\mathbf{R}_1 = \varnothing$

In the specification of the performance function of the TCS (cf. Table 9.1), P-facts are referred to by a particular value of the variable Cycle, for example, as in "phase (cycle1) = moving". In the specification of a contents base, only the variable is mentioned, as in $\mathrm{CB}(B_1) = \{\text{phase(Cycle)}\}$. To be complete, one should also add the value class for each variable. As an example, these are the value classes for the variables that are used in the case TCS:

phase(Cycle) : {moving, prolonged_moving, stopping, waiting}
move_time(Cycle) : $\mathbb{D}$
stop_time(Cycle) : $\mathbb{D}$
clear_time(Cycle) : $\mathbb{D}$

9.3 Elaborations

9.3.1 Specification of the PRISMA Model of Rent-A-Car

In this section, we use a slightly adapted version of the case Rent-A-Car (cf. Chap. 15) for illustrating the application of the PRISMA model to organisations. We will first discuss the white-box model and then the grey-box model.

9.3.1.1 The White-Box Model of Rent-A-Car

Figure 9.12 exhibits the prismanet diagram of a part of the Rent-A-Car organisation. It regards the settling of requests for concluding a rental contract, according to the PSI theory (cf. Chap. 8) and the generic transaction prismanet in Fig. 9.13.

The system consists of six elementary processors, eight single channels, three multiple banks, and one single bank. The interface with the environment consists of

the action channel C_1, the reaction channel C_2, the impediment channel C_3, the mutation bank B_1, as well as the multiple inspection banks MB_1, MB_2, and MB_3.

The processors outside the SoI, so the environmental processors, have been omitted, for the sake of simplicity. For the same reason, the revocation options are left out (cf. Chap. 8). Moreover, the multiple banks MB_1, MB_2, and MB_3 are connected through inspection links with the border of the SoI. This is a convenient way to express that they are inspection banks of all internal processors. Their content bases are respectively denoted as $CB(MB_1)$, $CB(MB_2)$, and $CB(MB_3)$.

From the diagram in Fig. 9.12, one can easily deduce the specification of the components **A**, **I**, **S**, **R**, and **M**, of the corresponding internal prismas. As an example, we provide the specifications of prisma1 (with processor P1 as its motor):

$\mathbf{A}_1 = TB(C_1) = \{$request ([rental] **is** completed)$\}$
$\mathbf{I}_1 = \varnothing$
$\mathbf{S}_1 \subseteq CB(MB_1) \cup CB(MB_2) \cup CB(MB_3)$
$\mathbf{M}_1 = \varnothing$
$\mathbf{R}_1 = TB(C_2) \cup TB(C_4) \cup TB(C_6) = \{$request ([rental] **is** paid), decline ([rental] **is** completed), promise ([rental] **is** completed)$\}$

Note. Without knowing precisely the performance function $\mathbf{P}_1$, we cannot be more specific about $\mathbf{S}_1$ than only stating that it is a subset of some other set. The reader is challenged to formulate $\mathbf{S}_1$ precisely after the specification of $\mathbf{P}_1$ is presented in Sect. 9.3.1.2.

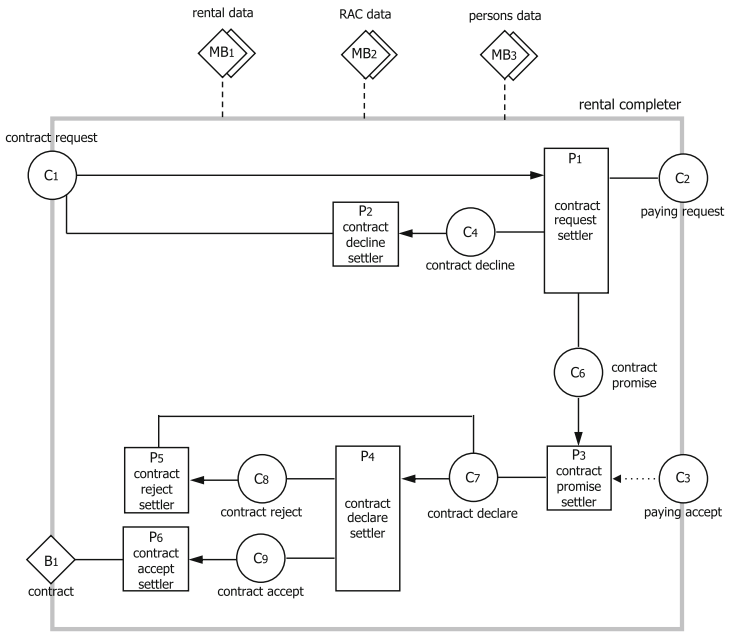

Fig. 9.12 Prismanet diagram of a part of the Rent-A-Car organisation

9.3.1.2 The Grey-Box Model of Rent-A-Car

On the basis of Fig. 9.12, we specify in Table 9.2 the performance function of prisma1 (with motor P1). As the specification language, we use a table form and a 'structured English' like language, which resembles the one that is applied in [14].

Table 9.2 Specification of prisma1 in the Rent-A-Car organisation

action	request ([rental] **is** completed) **with** starting_day[rental] : DAY ending_day[rental] : DAY renter[rental] : PERSON driver[rental] : PERSON car_group[rental] : CAR_GROUP pick-up_location[rental] : BRANCH drop-off_location[rental] : BRANCH
impediments	Ø
state	starting_day[rental] ∈ rental_horizon (year(starting_day [rental])) **and** ending_day[rental] ∈ rental_horizon(year (starting_day [rental])) **and** ending_day[rental] ≥ starting_day[rental] **and** duration[rental] ≤ max_rental_duration(year (starting_day [rental])) **and** #{cars **in** car_group[rental] **on** starting_day[rental] } > 0
mutation	Ø
reaction	promise ([rental] **is** completed) request ([rental] **is** paid)
action	request ([rental] **is** completed) **with** starting_day[rental] : DAY ending_day[rental] : DAY renter[rental] : PERSON driver[rental] : PERSON car_group[rental] : CAR_GROUP pick-up_location[rental] : BRANCH drop-off_location[rental] : BRANCH
impediments	Ø
state	starting_day[rental] ∉ rental_horizon (year(starting_day [rental])) **or** ending_day[rental] ∉ rental_horizon(year (starting_day [rental])) **or** ending_day[rental] < starting_day[rental] **or** duration[rental] > max_rental_duration(year (starting_day [rental])) **or** #{cars **in** car_group[rental] **on** starting_day[rental]} ≤ 0
mutation	Ø
reaction	decline ([rental] **is** completed)

The first action rule in Table 9.2 is performed when there is a request for completing a rental contract (first line of the action part). Note that, according to the PSI theory (cf. Chap. 8), there may be more than one request regarding the same rental in the course of time, but then it is a different event. Commonly, it will also have different properties (specified in the 'with' clause). There is no impediment for performing the first action rule, which is indicated by the symbol "Ø" in the impediments part. If the state condition is satisfied, so if its logical evaluation yields the value true, then the payment of the rental will be requested. The mutation is empty.

The state condition in the second action rule is the negation of the one in the first rule, whereas the action part and the impediments part are the same as in the first rule. If the state condition is satisfied, so if its logical evaluation yields the value true, then the concluding of the rental will be declined. Otherwise, nothing happens. The mutation is empty. Note that always either the first or the second action rule is executed successfully.

9.3.2 The Generic Transaction Prismanet

According to the PSI theory (cf. Chap. 8), C-acts/facts and P-acts/facts occur in universal patterns, called transactions. In Fig. 9.13, the complete transaction pattern from the PSI theory is reproduced. In order to express it in the PRISMA model, we need the prismanet whose corresponding diagram is shown in Fig. 9.14. It is called the *generic transaction prismanet*.

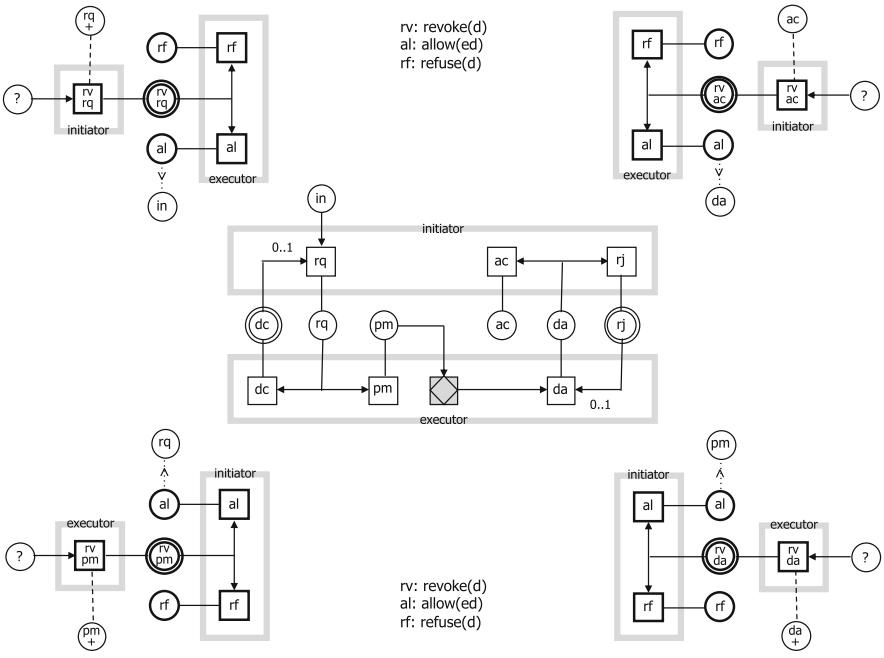

Fig. 9.13 The complete transaction pattern

There are nine processors that take care of the acts in the responsibility areas of the initiator in Fig. 9.13. They are labeled "Pin_1" through "Pin_9". Next, there are ten processors that take care of the acts in the responsibility areas of the executor in Fig. 9.13. They are labeled "Pex_1" through "Pex_{10}". The transmission bases of the channels are indicated by the abbreviated names of the intentions of the C-facts that they may transmit and contain: rq for request, pm for promise, etc. The four multiple channels, indicated with a "?", are (unknown) channels through which C-events are transmitted that trigger the connected performer to revoke [rv] one of the four basic C-facts in the transaction process: (rq), (pm), (st), and (ac).

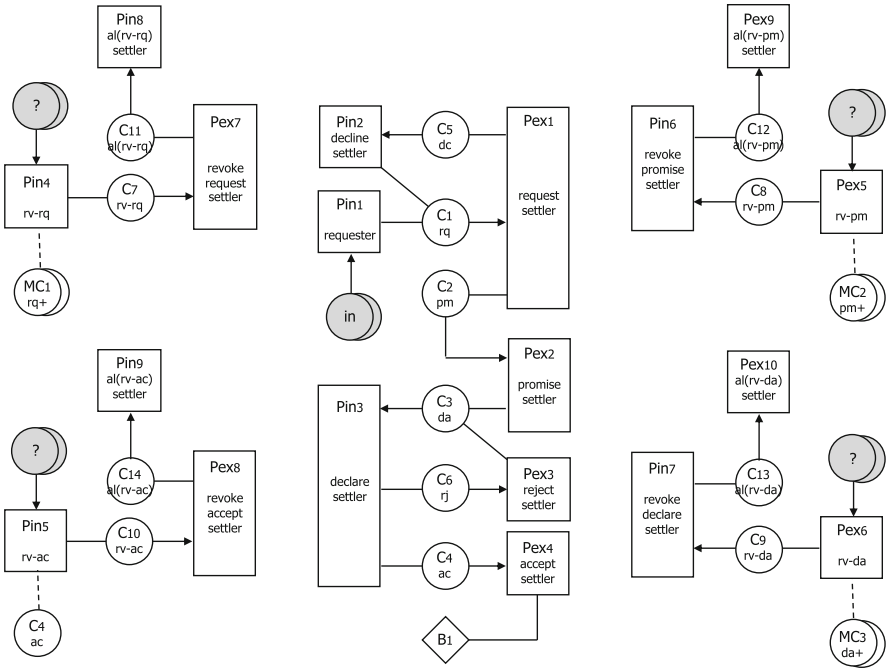

Fig. 9.14 The generic transaction prismanet

The specifications of the transmission bases of the three multiple channels MC_1, MC_2, and MC_3 are as follows: $TB(MC_1)$ is the union of $TB(C_1)$ through $TB(C_8)$; $TB(MC_2)$ is the union of $TB(C_2)$, $TB(C_3)$, $TB(C_4)$, and $TB(C_6)$; $TB(MC_3)$ is the union of $TB(C_3)$, $TB(C_4)$, and $TB(C_6)$. They correspond with the revocation conditions that hold for the complete transaction pattern. The contents base of bank B_1 consists of the independent P-fact type in the product kind, and all of its dependent P-fact types. The multiple channel, labeled "in", is included to complete the prismanet. Its transmission base consists of C-fact types to whose instances processor Pin_1 may respond by creating a request and putting it in channel C_1.

The operation of the generic transaction prismanet can briefly be explained as follows. If Pin_1 is triggered by an item in channel "in", it puts a request in channel C_1. In response, Pex_1 creates either a promise, put in channel C_2, or a decline, put in

channel C_5. In response to the decline, Pin_2 may create a renewed request, put in C_1. If this is not a feasible option, processor Pin_4 may create a [rv (rq)], in response to the decline (which is also contained in channel "?"), and put it in channel C_7.

Processor Pex_2 responds to a promise by creating a state fact, which is put in channel C_3. Processor Pin_3 responds to it by either an accept, put in channel C_4, or a reject, put in channel C_6. In the latter case, processor Pex_3 may respond by creating a renewed state, put in C_3. If this is not a feasible option, processor Pex_6 may create a [rv (st)], in response to the reject (which is also contained in channel "?"), and put it in channel C_9. In response to an accept in C_4, processor Pex_4 adds the corresponding independent P-fact and its dependent P-facts to the contents of bank B_1. This reflects the postulation in the PSI theory (cf. Chap. 8) that the product of a transaction is created at the moment that the accept fact is created. At the same time, this arrangement reflects the postulation that the executor of a transaction is the owner of the product, and thus the primary source to inquire about it. For each of the four revocation patterns, it holds that there is an unknown trigger in the channel that is indicated by "?", and that there is a wait event in channel MC_1, MC_2, MC_3, or C_4. The wait condition is that the status of the main process (the middle part of Fig. 9.13) must respectively be "requested or further", "promised or further", "stated or further", and "accepted". If the revoke of a request, promise, state, or accept is allowed, a corresponding C-fact will be put in respectively channel C_{11}, C_{12}, C_{13}, and C_{14}, which are action channels of respectively Pin_8, Pex_9, Pex_{10}, and Pin_9. These processors revert the main process to the statuses as indicated by the complete transaction pattern in Fig. 9.13. Note that if a refuse act is performed, the main process stays in the status it was in. No processors respond to refuse events. As said, ending up in a refuse state means that nothing has changed in the main process.

If the single processors Pin_1 through Pin_9 are combined in the composite processor CPin, and if the processors Pex_1 through Pex_{10} are combined in the composite processor CPex, one gets the prismanet as shown in Fig. 9.15 (left side). In this diagram, the channel names are left out, only the intention of the C-facts is mentioned. Moreover, the shapes of CPin and CPex are sinuated, in order to indicate that they are only shown partly, that is they may contain more components, because they are normally also connected to other prismanets (cf. Chap. 11). The use of composite processors, like the ones shown in Fig. 9.15, may be helpful in modelling discrete event systems, notably 'technical' systems that are actually technically implemented social systems. Examples of such systems are machines of all kinds, like vending machines, and control systems of all kinds, like warehouse control systems.

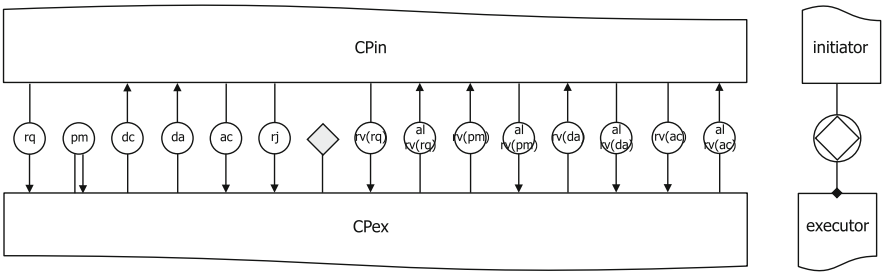

Fig. 9.15 Composite processors as organisational building blocks

On the right side of Fig. 9.15, the 'compression' of the left side part into the organisational building block (cf. Chap. 8) is shown. All channels (C_1 through C_{14}) and the bank B_1 are put together, resulting in the transaction-kind shape, which is a diamond (representing production) inside a disk (representing coordination). The small black diamond on the edge of the lower actor role shape indicates that it has the executor role in this transaction kind. Note that the executor role also comprises the processors, channels, and banks, that it needs as initiator in (other) transactions (if any), and that the initiator role also comprises the processors, channels, and banks, that it needs as executor of its 'own' transaction kind, as well as the processors, channels, and banks, that it needs as initiator in other transactions (if any). As already explained for the left part of Fig. 9.15, this is indicated by their sinuated shapes.

9.3.3 The C4E Quality Aspects

In this section, we will discuss the quality aspects of white-box systems in the PRISMA model, that is, of prismanets. To start with, we have shown that a prismanet is an *ontological* model of a discrete event system, that is, a white-box model that is fully abstracted from the (technological) implementation of the modelled system.

The first quality aspect of a prismanet is that it is *comprehensive*, which means that it is ontologically complete, of course provided that one has all knowledge of the concrete system. Consequently, it allows not only for studying the statics of the modelled system (its construction), but also its dynamics (its operation). Next, a prismanet is *coherent*, by virtue of the ontological system concept (cf. Sect. 9.2.1.2). It means that the model elements are connected in such a way that there are no 'loose' parts.

In addition, the presented prismanets are abstracted from realisation (cf. Chap. 11), so from all informational issues (like remembering, sharing, and deriving facts) and from all documental issues (like storing and retrieving documents or data). Note that derived facts just 'exist' in the ontological sense once they are defined. For example, someone's age at a particular day exists if the day exists and the person's birthday exists. The additional abstraction from realisation makes prismanets *concise*, by which we mean that their size is very small compared to current meta models, which mostly do not abstract from realisation and implementation. Moreover, prismanets are *consistent*, that is, they do not contain logical contradictions, as ensured by the PRISMA model.

The four quality aspects (coherent, consistent, comprehensive, and concise) constitute the requirements for calling the prismanet of a system its *essential model* (within the PRISMA model). By using the definite article "the", we want to conjecture that there is only one essential model for a given system. As a mnemonic, the quality requirements, together with their corollary of capturing the essence of a system, are collectively named "C4E". The added connotation is "see for E", expressing that one must always strive to capture the essence of a system, in order to reduce the complexity of its white-box model, and consequently to get deeper insight into and better overview over the system. The reduction of complexity that is

achieved by producing a white-box model that satisfies the C4E requirements, contributes to achieving the generic enterprise engineering goal of intellectual manageability [15].

9.4 Discussions

9.4.1 Implications of the DELTA Theory for Software Engineering

Software engineering is the discipline of engineering software systems. It includes the design, the implementation, the deployment and the maintenance of these systems. Even professional software engineers sometimes seem to forget that a software system in operation is a mathematical machine (also if its function is to support people in organisations). This undeniable truth has important implications for the discipline of software engineering, however.

First, it implies that a software engineer must have a *comprehensive* understanding of the object system that is going to be supported by the software system that he/she is going to develop. The relationship between the object system and its supporting information/software system is precisely defined in the ALPHA theory (cf. Chap. 11). Moreover, this understanding of the object system must be *concise* (thus fully abstracted from realisation and implementation) in order to manage intellectually its complexity, that is, to get and keep insight into and overview over of the software system to be developed. This insight and overview is also indispensable for *validating* that the software system satisfies the applicable requirements.

Second, such an understanding of the object system is also indispensable for *verifying* the logical correctness of the developed software system. The role of testing can only be secondary, because it may show the presence of errors but never their absence, as pointed out already long ago by Edsger Dijkstra [16].

Third, such an understanding would also be a necessary basis for studying the construction and operation of the system by means of mathematical and logical analysis, as well as through simulation, possibly including animation.

Fourth, such an understanding of the object system would be a necessary condition for generating software in such a way that its correctness can be guaranteed [17]. Because PRISMA models are formalised, they can be converted to mathematical/logic complexes (cf. Chap. 6), and subsequently expressed in a programming language.

9.4.2 Prismanets and Petri Nets

Readers who are familiar with Petri nets may have wondered already what the similarities and differences are between prismanets and Petri nets [18], because the

resemblance of the graphical symbols suggests some similarity. It seems worthwhile therefore to make a comparison of the two process modelling techniques.

In [19], the relationship between the Petri net and the smartienet is investigated. Since the smartienet [20, 21] is the precursor of the prismanet, we will first summarise the findings in [19], using the TCS again as the example system. The smartienet diagram in Fig. 6 in [19] is reproduced as Fig. 9.16. The legend of the diagram is the same as for the prismanet diagram (cf. Fig. 9.10). So, there are two processors (P_1 and P_2), four banks (B_1, B_2, B_3, and B_4) and two channels (C_1 and C_2). Because the smartienet lacks the notion of delayed mutation, which the prismanet does have (cf. Sect. 9.2.3.2), two elementary processors are needed to model the TCS properly. Next to being activated by set_phase commands from P_1, P_2 also activates itself, through channel C_2.

Although the TCS is a quite simple system, its discussion above and in [19] illustrates how difficult it is to understand a system comprehensively (i.e. its construction, its operation, and the effects of its operation, thus the processes or state trajectories in the system's world) without a proper theory. At the same time, both the prismanet model in this chapter and the smartienet model in [19], demonstrate the power of ontological modelling: providing one with a comprehensive, coherent, and consistent understanding of a system, released from the burden of implementation details.

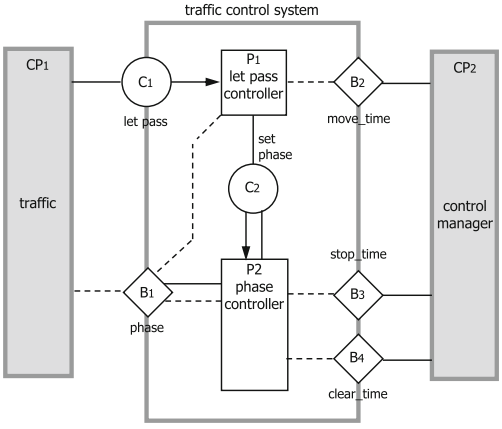

Fig. 9.16 Smartienet diagram of the TCS

The Petri net [18] originates from the research work that Carl Adam Petri[5] undertook already at a young age. It can best be understood as a (meta) model for studying synchronisation problems in discrete event systems. A well-known application of the Petri net is the studying of cooperating sequential processes in computers. Because of its popularity among (business) process modellers with a focus on

[5] https://en.wikipedia.org/wiki/Carl_Adam_Petri

formalisation, the Petri net has evolved in the course of time in order to meet the additional requirements. The major extensions have been the addition of delay times (resulting in the timed Petri net [22]) and the 'colouring' of tokens, by which is meant adding process semantics (resulting in the coloured Petri net [23]).

In Figs. 9.17 and 9.18 the timed Petri net diagram of the TCS is exhibited. It is an adapted version of Fig. 2.10 in [24]. The adaptation comprises a more accurate layout and corrections of the delay times. The disks represent the places of the Petri net and the boxes represent the transitions. The black dots in the places are the tokens that can 'move' through the net. A place is an input place of a transition if its shape is connected to the transition shape by an arrow. A place is an output place of a transition if its shape is connected to the transition shape by a solid line. Transitions 'fire' if all input places contain at least one token. The effect of 'firing' is that one token is removed from every input place and that one token is added to every output place. The names of the places and transitions, as well as the time delays, correspond with the names in Fig. 9.8. So, for example, W1 means Cycle 1 is in the state waiting, S1–W1 is the transition from S1 to W1, and CT2 is the clear time in Cycle 2.

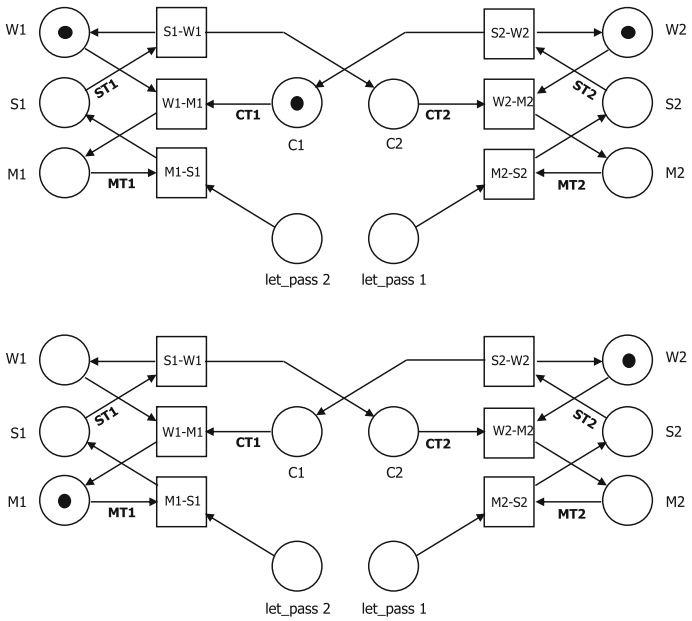

Fig. 9.17 Timed Petri net of the TCS (1)

The process starts in the situation that the state of Cycle 1 is waiting and the state of Cycle 2 is (prolonged) moving (cf. Fig. 9.17 upper part). Then a let_pass event in Cycle 1 occurs (a token is put in place let_pass 1). The condition to fire M2-S2 is now satisfied, and thus the transition takes place. The state of Cycle 2 will immediately become stopping (cf. Fig. 9.17, lower part). This enables the firing of S2-W2

after ST2 time units. Then the token in S2 is removed and a token is added to both W2 and C1 (cf. Fig. 9.18, upper part). Together with the token in W1, this satisfies the condition for W1-M1 to fire, after CT1 time units. The effect of this transition is that the tokens in W1 and C1 are removed and that a token is added to M1, meaning that the state of Cycle 1 will become 'moving' (cf. Fig. 9.18, lower part). After MT1 time units, transition M1-S1 is enabled by this token, but the firing has to wait for a token in place let_pass 2, so for arriving traffic in Cycle 2. As soon as this is the case, the whole process will be repeated, but now with the cycles reversed.

Despite the similarity of the diagrams, the Petri net that is represented in Figs. 9.17 and 9.18 and the prismanet that is represented in Fig. 9.11, are fundamentally different. The prismanet is a *construction model*: it shows the composition, the environment, and the structure of a system (cf. Sect. 9.2.1.2). The Petri net, however, is a *process model* in the strict systemic sense [4]: it specifies the lawful states and the lawful transitions of a system's world, possibly the world of the system that is represented by the prismanet diagram in Fig. 9.11. The Petri net in Figs. 9.17 and 9.18 is semantically equivalent to the STD in Fig. 9.8. Note that the presented Petri nets are independent of any implementation, contrary to the one in [24]: using red, yellow, and green lights to inform traffic participants about the states 'waiting', 'stopping', and 'moving', is just one way of implementing the inspection link from the processor 'traffic' to the bank 'phase' in the prismanet diagram in Fig. 9.11. Such implementation choices should not appear in an ontological model.

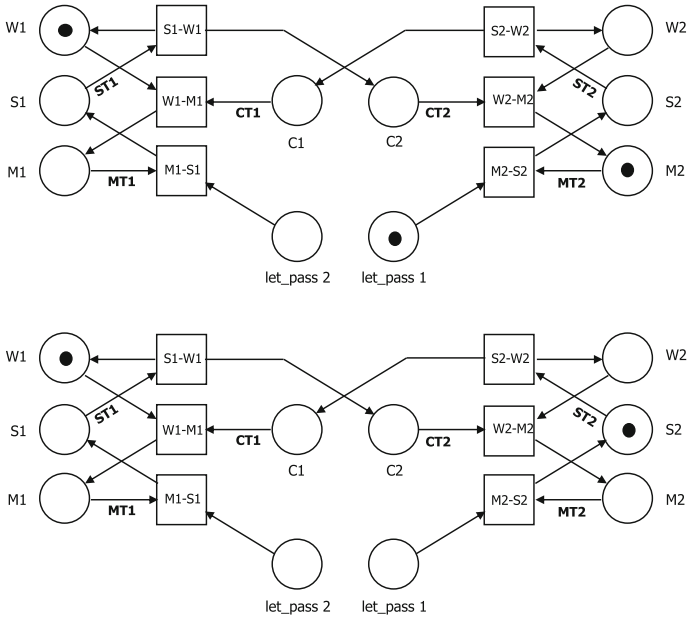

Fig. 9.18 Timed Petri net of the TCS (2)

9.4.3 The Petri Net and the DEMO Process Model

Because Petri nets are process models, as discussed in the previous section, it is interesting to compare the Petri net with the DEMO Process Model (PM), as presented in Chap. 12. For some time already, Petri nets have been applied for analysing and simulating (business) processes that are modelled in DEMO, for example, in [25] and in [26]. As discussed in Chap. 8, DEMO process models are basically tree structures (cf. Chap. 10) of transaction processes. So, let us first draw a comparison between the Petri net and the complete transaction pattern (CTP), by interpreting the CTP as a Petri net.

Figure 9.19 contains the CTP from Fig. 9.13, but without the names of the C-facts, in order to let them resemble the places of a Petri net. The C-acts must be interpreted as transitions. The optionalities (indicated by the cardinality ranges 0. . .1 next to the response links in Fig. 9.13) are accommodated by adding the alternatives (cf. Chap. 8): performing rv-rq and rv-da, respectively. Moreover, the P-act and the P-fact are separated: the P-fact symbol (the diamond) is also interpreted as a place in the Petri net. The reversion states are indicated in the same way as they are in Fig. 9.13, in order to avoid crossing lines in the diagram.

Let us suppose that there is a token in the place labeled "in". Then transition rq will fire, resulting in putting a token in its output place (which corresponds with the state requested). Then both transition dc and transition pm are enabled. However, only one of them can fire because there is only one token in the input place. Going on in this way, one will discover that the process either ends (successfully) in the state accepted or goes on infinitely, which is exactly the idea of the CTP (cf. Chap. 8).

What is new in Fig. 9.19, compared to the 'real' Petri net, are the dashed lines. In a Petri net, they have to be replaced by solid arrows, and thus serve as an additional input place for the four revocation transitions, because state conditions and response conditions cannot be distinguished in a Petri net. In other words, the CTP (and for that matter the DEMO PM) is richer than a Petri net as regards the ability to represent real (business) processes, where the distinction between state conditions (inspection links) and triggers (response links) is crucial to deeply understand them.

A similar remark can be made with respect to the wait links in the DEMO PM (cf. Chap. 12). They are also crucial in real business processes: waiting for something to happen before acting, is fundamentally different from being triggered to act.

As said, a Petri net only represents the world of a system, not the system itself: it is (only) a grey-box model, not (also) a white-box model. To understand the corresponding system, one has to model it in some other way, for example, as a prismanet.

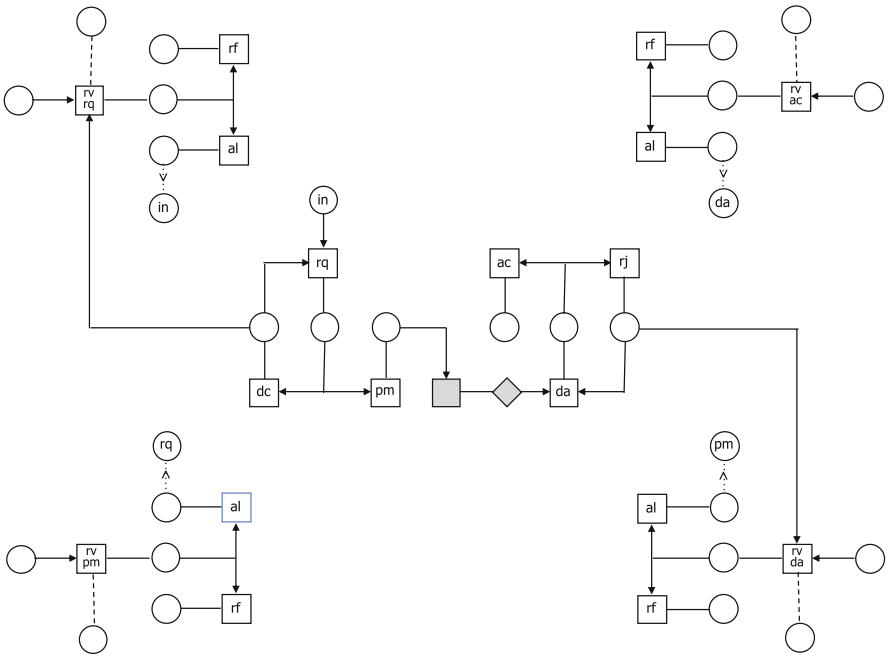

Fig. 9.19 Petri net interpretation of the CTP

The current use of (timed coloured) Petri nets for modelling business processes is a typical illustration of how meta models can be 'inflated' to accommodate applications where they were never meant for (and consequently are mostly not suited for). As convincingly discussed in Chap. 8, business processes are processes that occur in the coordination world of organisations, as the effect of acts by autonomous human actors. Other examples of 'inflations' of modelling approaches are the use of the Entity Relationship Diagram (originally meant for exhibiting the structure of relational databases) for conceptual modelling, and the use of the UML class diagram for conceptual modelling and even for ontological modelling (cf. Chap. 6).

With reference to Einstein's quote at the beginning of Chap. 4 (*Whether you can observe a thing or not depends on the theory that you use. It is the theory that decides what can be observed*), all modelling actions are inherently shaped by the theory one applies or, when lacking an explicit theory, by the 'mental glasses' one has put on. If these glasses are ill-suited, the resulting models will not be very useful. Unfortunately but most likely, one may not be aware of the mismatch.

References

1. Bertalanffy, L. V. (1969). *General system theory; foundations, development, applications* (Rev. ed., xxiv, 295 p.). New York: G. Braziller.
2. Wiener, N. (1965). *Cybernetics: or, Control and communication in the animal and the machine* (2nd ed., 212 p.). The M I T paperback series. Cambridge, MA: MIT.

3. Legasto, A., Forrester, J. W., & Lyneis, J. M. (1980). *System dynamics*. *TIMS studies in the management sciences* (282 p.). Amsterdam: North-Holland.
4. Bunge, M. (1979). Treatise on basic philosophy ontology II: A world of systems. In *Treatise on basic philosophy 4* (p. 1). Dordrecht: Springer.
5. Weinberg, G. M. (1975). *An introduction to general systems thinking* (xxi, 279 p.). Wiley series on systems engineering and analysis. New York: Wiley.
6. Marquis, J.-P. (1996). A critical note on Bunge's 'system boundary' and a new proposal. *International Journal of General Systems, 24*(3), 245–255.
7. Bunge, M. (1977). Treatise on basic philosophy ontology I: The furniture of the world. In *Treatise on basic philosophy 3* (p. 1, 370 p.). Springer: Dordrecht.
8. Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2010). *Feedback control of dynamic systems* (xviii, 819 p., 6th ed.). Upper Saddle River, NJ: Pearson.
9. Marca, D., & McGowan, C. L. (1988). *SADT: Structured analysis and design technique* (xvii, 392 p.). New York: McGraw-Hill.
10. Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems* (xxiii, 769 p., 2nd ed.). New York: Springer Science + Business Media.
11. Hoogervorst, J. A. P. (2017). *Foundations of enterprise governance and enterprise engineering* (p. 574) Cham: Springer International.
12. Aveiro, D., Silva, A. R., & Tribolet, J. (2010). Towards a G.O.D organization for organizational self-awareness. In *6th International Workshop, CIAO! 2010*. St. Gallen: Springer.
13. Hee, K. M. V., Houben, G.-J., & Dietz, J. L. G. (1989). Modelling of discrete dynamic systems; framework and examples. *Information Systems*, *14*.
14. Perinforma, A. P. C. (2015). *The essence of organisation*. South Holland: Sapio Enterprise Engineering.
15. Dietz, J. L. G., & Hoogervorst, J. A. P. (2013). The discipline of enterprise engineering. *Journal Organisational Design and Engineering, 3, 28*.
16. Dijkstra, E. W. (1970). *Notes on structures programming*.
17. van Kervel, S. J. H., et al. (2012). Enterprise ontology driven software engineering. In *ICSOFT 2012*. SciTePress.
18. Peterson, J. L. (1981). *Petri net theory and the modeling of systems* (x, 290 p.). Englewood Cliff, NJ: Prentice-Hall.
19. Dietz, J. L. G. (2005). System ontology and its role in system development. In J. Castro & E. Teniente (Eds.), *Advanced information systems engineering wokshops* (CAiSE) (pp. 271–284). Porto.
20. Dietz, J. L. G. (Ed.). (1987). *Modelleren en specificeren van informatiesystemen*. Eindhoven: T.N. Eindhoven University of Technology.
21. Houben, G.-J., Dietz, J. L. G., & van Hee, K. M. (1988). The SMARTIE framework for modelling discrete dynamic systems. In P. Varaiya & H. Kurzhanski (Eds.), *Discrete event systems: Models and applications*. New York: Springer.
22. Wang, J. (1998). *Timed petri nets – Theory and applications*. Boston: Kluwer Academic.
23. Jensen, K. (1997). *Coloured petri nets: Basic concepts, analysis methods, and practical use* (2nd ed.). Monographs in theoretical computer science. Berlin: Springer.
24. van der Aalst, W., & van Hee, K. M. (2004). *Workflow management models, methods, and systems. Cooperative information systems series* (368 S.). Cambridge, MA: MIT.
25. Barjis, J. A., & Dietz, J. L. G. (2001). A type of petri net based on speech act theory for modeling social systems. In A. W. Heemink (Ed.), *EUROSIM*. Delft.
26. Barjis, J. A. (2008). The importance of business process modeling in software systems design. *Elsevier Science of Computer Programming, 71*, 73–87.