5 Representing state and behaviour in MWDS

As described in the introduction to the thesis, this chapter sets out to illustrate how MWDS can be viewed (controversially) as providing an alternative and unorthodox picture of the relationship between the world and our representation of it. In the experience of the author, such an alternative viewpoint is natural for the modeller engaged in MWDS. This leads the author to conceive the distinction between MWDS and traditional computer-based modelling (for example, in respect of openness, and experiential character) as fundamental rather than as a matter of degree. This is potentially a source of controversy for sceptical readers – one that becomes even more pronounced in Chapter 6, where such a fundamental distinction is presumed.

Representing state is a key concern in MWDS. The discussion in this chapter addresses the connection between state and behaviour in MWDS. By introducing the concept of an ADM device and a program device, it is possible to explore the link between MWDS and traditional modelling of state and behaviour. The ADM device and the program device are compared and contrasted with reference to four aspects of state – Situational, Explicit, Mental and Internal – that are of concern when we use and interact with a device. The two principal case studies in this chapter relate to:

- a simple program (‘Jugs’), originally developed for the BBC micro-computer by R. Townsend [Town], that has been widely used in schools;
- a CSP specification for a chocolate vending machine introduced by C. A. R. Hoare [Hoare85],

which are respectively used to explore the explicit and internal, and the situational and mental aspects of state.

5.1 Open-development and closed-world views on devices

Traditional programming, whether declarative or procedural, focuses on the representation of actions and behaviours. State is defined with reference to behaviour. MWDS is not like traditional programming. It is more closely related to building a physical artefact. A definitive script itself represents a state not behaviour. Behaviour in a definitive context is understood to refer to a predictable, reliable pattern of state changes that can be repeated many times. When we construct a definitive model, behaviours are associated with patterns of interaction that occur at the discretion of the modeller. However, there may be no behaviours associated with definitive scripts that are developed with exploratory creative model building in mind.

To compare a definitive model with a traditional program, we need to make an ontological shift, whereby we regard a program as it executes on a computer as a physical artefact that generates observables that can be experienced by the user. Even after this shift in perspective, a definitive model is quite unlike a traditional program, which is designed with a specific function in mind. Such a program reflects a 'closed-world' model of the requirement; it is concerned with behaviour and interactions which are circumscribed and preconceived in advance by modellers who have a good understanding of what users want. In this sense, the term 'device' is used to refer to a physical artefact that is conceived as 'program-like' (whether or not it is computer-based). In particular, we shall refer to a traditional program that is being viewed as a physical artefact as a 'program device'.

The characteristics of a typical device are listed in the first column of Table 5-1. There are specific modes of use within which the behaviour of the device and of its user follow standard patterns and the interpretation of state changes is preconceived. These can be described in terms of the 'circumscribing observables' that characterise its use. These circumscribing observables may refer to internal and external features of the device and to factors associated with the situation and the user's understanding of the situation. The set of circumscribing observables defines a boundary surrounding normal use of the device. Since the device is designed for general rather than merely personal use, the set of circumscribing observables is defined in an abstract and objective manner.

Characteristic	**Typical devices**	**ADM artefacts**
Specific modes of use	✓	✗

Standard patterns of behaviour	✓	✗
Standard user interaction patterns	✓	✗
Standard interpretations for state change	✓	✗
Clear identifying boundary	✓	✗
An objective observational context	✓	✗

Table 5-1: Contrasting the characteristics of typical devices and ADM artefacts

Table 5-1 contrasts the characteristics of devices and ADM artefacts. Though the characterisation in the table is flexible, it reflects a genuine distinction. The characteristics of a device cannot be completely circumscribed. It has properties that are not considered in standard use and it can be used in ways outside the scope of its specification. For instance, an analogue watch can be used as a paperweight or used in conjunction with the sun for direction finding. As illustrated in Chapter 3, ADM artefacts can have all the characteristics specified in Table 5-1. Such examples illustrate the shift from 'closed' user-artefact interaction (i.e. interaction conforming to the user manual) to 'open' user-artefact interaction (i.e. adaptive use involving interaction with the artefact not anticipated by the designer) and vice versa (cf. [BRWW01]). The shift from closed to open interaction relies upon the user's imagination. The shift from open to closed interaction relies upon the modeller's discretion. A device is constrained from being open by optimisation and its embodiment in the physical world. An ADM artefact cannot be optimised to a specific function without compromising its quality as a construal. How efficient a device can be depends on what physical properties we can identify in the world. How flexible an artefact can be depends on how faithfully we can construe its interaction with the world.

From an abstract viewpoint, the distinction between devices and ADM artefacts resembles the distinction between a traditional objectivist view of cognition and experiential cognition, as described by Benyon and Imaz [BI99] (after Lakoff [Lakoff88]) in connection with their research into the conceptual foundations of representations used in HCI and software engineering:

> "… experiential cognition emphasizes the role of both bodily and sociocultural experience in characterizing concepts and in the human imaginative capacity for creating concept and modes of rationality that go well beyond any "mind-free" external reality."

> "Experiential cognition is an approach to understanding what meaning is to humans. The traditional objectivist view of cognition sees it as the algorithmic manipulation of abstract symbols that provide internal representations of an external reality."

It also reflects the distinction between the sense-making activities that are attributed to these two views of cognition as explained in [BI99] as follows:

> "Traditional views of cognition see meaning coming from the association of symbols with external objects, whereas experiential cognition sees meaning coming from the application of "imaginative projections" to some basic concepts, these basic concepts being meaningful because of their roles in bodily experience."

The traditional objectivist view of cognition is similar in spirit to the traditional view of software engineering: when we begin to build a device, we go through a process of negotiation between users' requirements and software limitations which is generally done through a paper-based design. In contrast, experiential cognition has more in common with experimentally-based design which directly builds a prototype for a device, then modifies and refines it.

The discussion in Chapter 3 has shown that ADM artefacts are very rich and broad. In modelling a device, they can take account of all possible aspects, including the internal factors, such as mechanisms and interfaces, and external factors, such as the situation and the user's mind. However, in order to make the comparison between MWDS and traditional ways of studying the development and use of devices possible, we need to restrict the observation and interaction with an ADM artefact so that it has the characteristic properties of devices identified in Table 5-1. The term 'ADM device' will be used to refer to an ADM artefact that is restricted in this way by exercising the modeller's discretion.

In the rest of this chapter, we take up the two agendas suggested by the previous paragraph: a comparison between the internal construction of traditional devices and ADM devices; and the application of ADM artefacts in studying the design and use of traditional devices in their broader context, taking account of mental and situational factors. These two agendas respectively relate to the internal and the external semantic relations associated with a device as depicted in Figure 2-12.

Four aspects of state in the study of devices

These two agendas above are discussed with reference to four aspects of state – **S**ituational, **E**xplicit, **M**ental and **I**nternal – that are defined based on the way that we use or interact with a device (cf. Figure 5-1). As discussed in [BRWW01], these four aspects are relevant in both closed user-artefact interaction and open user-artefact interaction. They are specified as follows:

- **E**xplicit state corresponds to the visible state of the device;

- **S**ituational state is related to knowledge of the real-world context in which the device is being used and to which the device refers;
- **M**ental state takes account of the user's knowledge and expectations about interaction with the device when interpreting its current state and conceiving its possible transitions of state;
- **I**nternal state is related to the internal representation or mechanism inside a device. In normal use, it may not have to be considered unless or until we realise that there is a conflict in the relationship between the device and its referent.

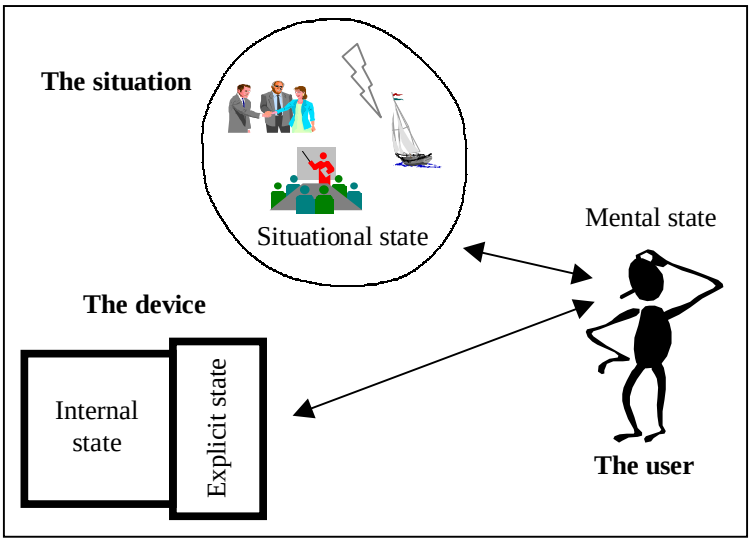


Figure 5-1: The four aspects of state in using a device (adapted from [BRWW01])

As in [BRWW01], these four aspects of state are illustrated with reference to a digital watch. Explicit state is what we can see by looking at the watch. If we observe the explicit state of a device in isolation from other aspects of state, we may misinterpret the state of the device. For instance, by looking at the watch, it is sometimes hard to tell whether we are observing the current time (i.e. if the watch is in 'display current time' mode) or the alarm time (i.e. if the watch is in 'display alarm time' mode). The situational aspect of state typically supplies the norm for device use. For instance, when the user sets the watch to 'stopwatch' mode to record the time taken to run a race, he/she has to observe features of the external situation in addition to observing the watch. Mental state refers to knowledge complementary to the explicit state that has to be carried in the user's mind to make sense of the device's behaviour. For instance, the user expects that if he/she presses a button X once in the current time mode, the watch will display the alarm time that is currently set. The internal state of the digital watch is concerned with test and repair. The internal state of the watch is not usually accessible to the user.

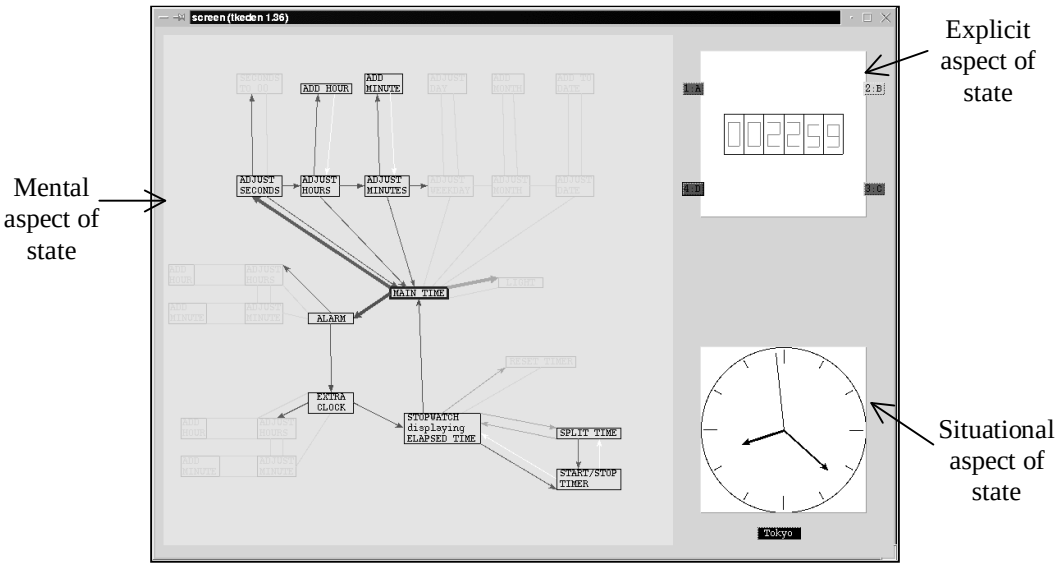


Figure 5-2: Situational, Explicit and Mental aspects of state in digital watch use (adapted from [BRWW01])

The three primary aspects of state (viz. Explicit, Situational and Mental) are typically specific to each instance of use, and have to be simultaneously apprehended by the user [BRWW01]. For instance, when using a watch to count pulse rate, the user puts the watch into the countdown mode (registered as mental state), sets the duration to one minute (consulting the explicit state) and counts pulses (referring to the situational state) until the watch beeps (establishing an explicit state and confirming the expectation associated with the mental state).

A device is normally designed with a concept of appropriate use in mind. This will be framed in the 'user manual' for the device in terms of idealised observables related to the primary aspects of state. For example, a digital watch user who has just arrived in Tokyo from London will need to know how to reset the time on the watch to Japanese time. The designer expects the user to be able to read the current time on the watch (an explicit state), to know the time in Tokyo (a situational state) and to have some recipe in mind (a mental state) for changing the watch to an appropriate mode (cf. Figure 5-2). The uses of a device are conceived and documented with reference to common types of abstract observation relating to explicit, situational and mental state that are determined by the nature of the device. A prescription for use in terms of abstract observation is not in itself sufficient for practical situated use. For instance, a digital watch is designed to be used in different time zones and this kind of use will be documented in the user manual with reference to the current time (an 'abstract' observable), but this does not tell the user

how to determine the time in Tokyo. There are two ways in which ADM artefacts can be used to study devices:

- **constructing ADM artefacts that model devices**: An example of such use is depicted in Figure 5-2, where the model of digital watch is as discussed in Chapter 3. Another example is the model of the original Jugs program [Town] depicted in Figure 2-5, which can also be interpreted as a model of a device. This use of an ADM artefact focuses on internal and explicit aspects of state associated with the internal semantic relation (cf. Figure 2-12(a)). This is discussed in more detail in Section 5.3.1 with reference to the relationship between MWDS and traditional programming techniques such as data structures, flowcharts and the Model-View-Controller architecture.
- **developing an ADM artefact to support the construction and to explore the situated use of a device**: Examples of such development are depicted in Figure 5-2 and Figure 5-5, which respectively model the situated use of a digital watch and the Jugs program. This use of an ADM artefact focuses on mental and situational aspects of state associated with the external semantic relation (cf. Figure 2-12(b)). This is discussed in more detail in Section 5.3.2 with reference to the relationship between MWDS and specification techniques such as the use of CSP, use-case analysis and the Z-notation.

In thinking about representations for interactive systems design, it is usual to make a distinction between *explicit* representations, such as models and artefacts, and *implicit* representations that involve interpretation on the part of the user, and rely upon assumptions about his/her background, role and culture [BI99]. This distinction echoes the traditional distinction between what is viewed as empirically given and what is viewed as the product of rational construction. In MWDS, as explained in Sections 1.6 and 2.1, the association between the model and its referent is mediated experientially. What matters in establishing this association is what is observable – that is, what can be directly apprehended by the modeller – whether or not this apprehension involves interpretation. By this criterion, features of an ADM artefact that require sophisticated understanding on the part of the modeller (such as the English annotations on the buttons in the Jugs model in Figure 2-5, the graphical presentation of the months of the year in Figure 2-9 and the Thai transliterations of numbers in Figure 2-16) are potentially regarded (depending on the modeller) as part of the explicit state. On this account, MWDS can be seen as involving a shift in perspective on what is empirically given similar to that proposed by William James in his 'Radical Empiricism' [James96, Bird86].

The special qualities of an ADM artefact in representing devices in design and use stem directly from the fundamental role played by the observable. In MWDS, 'what can be directly apprehended by the modeller' is the central focus for attention and exploration. Through experience, 'what can be directly apprehended by the modeller' can evolve as what at first requires 'off-line' interpretation becomes immediate and explicit. Through projection and experiment, the modeller can identify what is – as if – directly apprehended by other agents. The implications of this are that modelling a device can potentially take account of failures and defects, and that modelling the use of the device can go beyond the designer's stereotype of idealised use to take account of richer real-world observables from the situations and subjective characteristics of the users, both separately and in combination.

5.2 The SEMI aspects of state in MWDS

For a device, the way that SEMI aspects of state are related is precise and circumscribed to reflect the designer's abstract conception of its use. Users normally interact with the device in a precise way based on the written manual. Closed-world modelling of the device only supports interaction that is rigid and bounded. This is because it is built and optimised to meet a precise and predefined specification. In open development of an ADM device, the possible interaction is more open-ended and gives flexibility to adjust and refine the characteristics of the device and its users. Because the boundary of an ADM device is established by the modeller's discretion, it still has interactive and experience-based characteristics. This gives scope for richer exploration of the SEMI aspects of state associated with the device.

The four aspects of state and their different qualities are discussed and clarified in this section with reference to extensions of the Jugs model as depicted in Figures 2-5 and 3-6. For this purpose, imagine (Scenario 1) that the original Jugs program [Town] is being used by pupils in a standard way. From the initial state depicted in Figure 5-3, the objective for the pupil is to get one unit of liquid by a sequence of jug operations. For instance, a suitable sequence of operations is:

Fill A; Pour; Fill A; Pour; Empty B; Pour; Fill A; Pour.

The teacher is able to specify a new problem by changing the capacities of the jugs and the target.

Internal and Explicit aspects of state

The internal and explicit aspects of state are directly associated with the physical characteristics of a device. The designer's conception of a device refers to observables – possibly abstract rather than realistic – that represent its internal state. In developing a definitive model of a device, there are counterparts for these observables in the script. Normally the internal state should be hidden from the users. It can be interpreted as **state that is recorded** inside the device.

Figure 5-3 represents the Jugs model viewed as an ADM device ('the Jugs device'): if we interact with the model in a restricted way (i.e. by pressing the interface buttons), it serves precisely the same function that an implemented program meeting the specification for 'jugs' does [Town]. We can compare the use of the Jugs device in Scenario 1 with that of the Jugs program.

Where the Jugs program is concerned, the internal state is recorded in procedural variables. In contrast, the internal state of the Jugs device is recorded in the extract of the script labelled Internal in Figure 5-3. Such an internal representation is closely related to the designer's view of the internal state of the Jugs program. In the Jugs device, there is scope to change the internal state in a way that does not affect the explicit state of the model or the functionality of the model that is exercised in Scenario 1. For instance, if the teacher wishes to supply new values for capA and capB so that the target 1 is still appropriate, the modeller can introduce these definitions:

```
n is 2*k+1; capA is 5*n; capB is 5*n+2; k=0;
```

Alternatively, to ensure that randomly chosen values of capA and capB still lead to a solvable problem, the modeller can introduce the single definition:

```
target is gcd(capA, capB);
```

These show the flexibility for adapting the Jugs device for more specialised use and to enrich its internal mechanism.

Note that someone who explores the Jugs device (cf. Figure 2-10) may not be aware of internal changes of this kind until he/she hits upon the right interaction (e.g. redefining k). For the explorer, understanding the semantics of the Jugs device involves privileges to interact associated with an expectation that may or may not be realised.

The explicit state of a device is directly connected to the visualisation or appearance of the device. It corresponds to **state as perceived** from the viewpoint of an external observer. Users always apprehend a device from its explicit state. For example, the explicit state of the Jugs

device is depicted in Figure 5-4. When both jugs are initially empty (cf. Figure 5-3), and the user clicks on the Fill B option, the explicit state of the Jugs device is changed to the state depicted in Figure 5-4(a). In fact, this click invokes the internal agents init_pour and pour to change the value of the internal observable contentB, which sequentially affects the explicit state of the Jugs device.

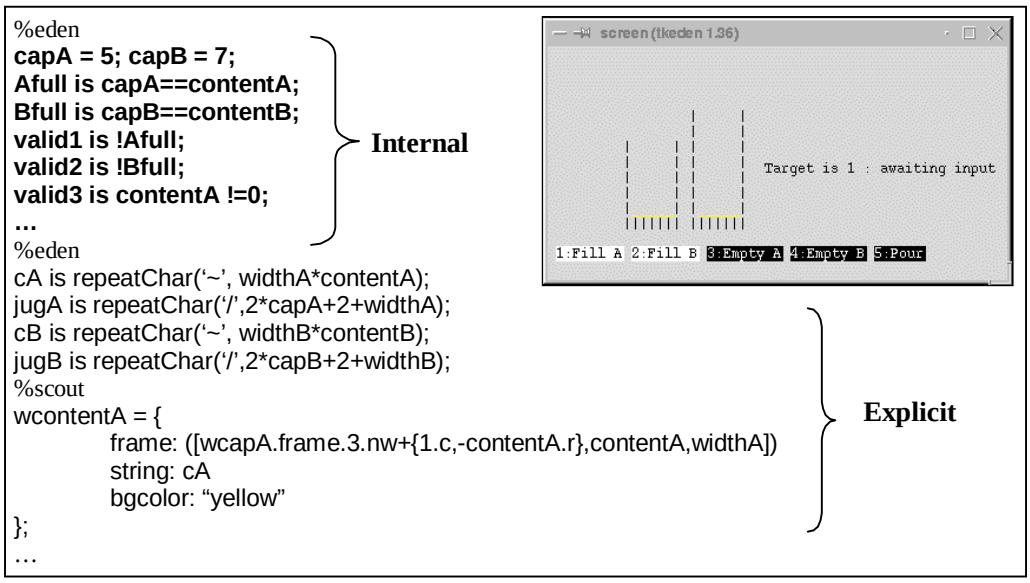


Figure 5-3: A screenshot of the Jugs device with its script (for internal and explicit)

The dependency between the explicit and the internal state of the model is established by the definitive script labelled Explicit in Figure 5-3. Changing the internal observable capB automatically affects the explicit state of the model as depicted in Figure 5-4(b).

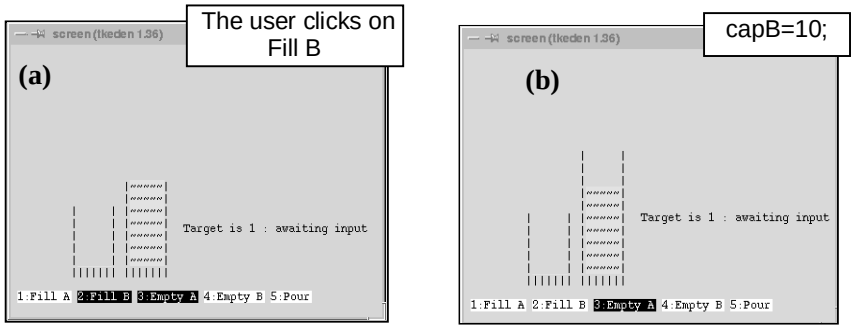


Figure 5-4: Explicit state for the Jugs device

Note that in the script in Figure 5-3, the colour of the liquid is treated as a component of the explicit state that is defined independently of the internal state of the model. This can be interpreted as meaning that the colour of the liquid is not a circumscribing observable of the device: assigning a new colour to the liquid does not affect the functionality of the device. This

illustrates the designer's idealised view of the circumscribing observables, which presumes that the user sees the level of the liquid regardless of its colour (cf. Figure 3-3 where no colour is involved).

Mental and Situational aspects of state

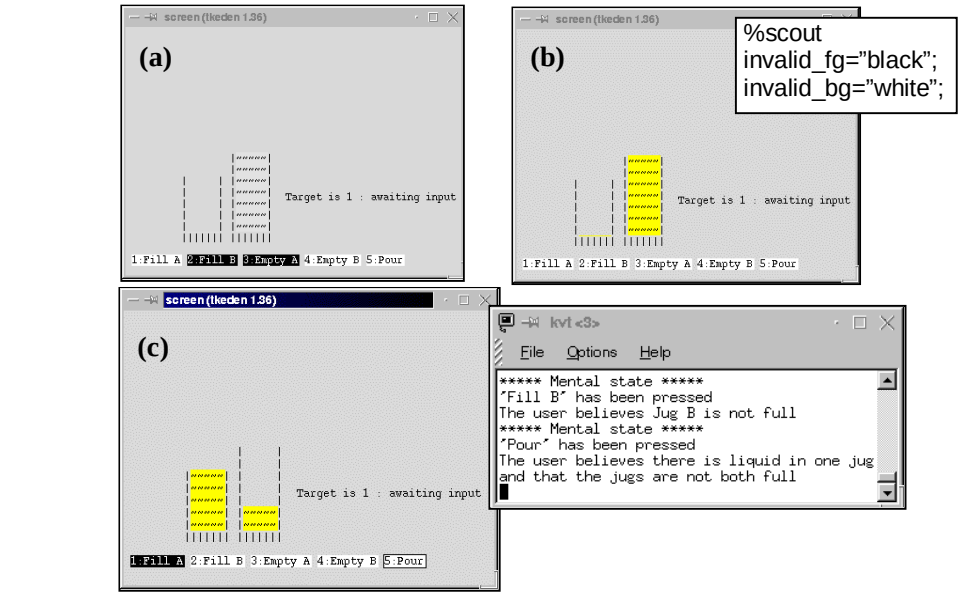
The representations of the internal and explicit aspects of a device are objective in nature. They reflect directly the designer's conception of the device and its identifying boundary (cf. Table 5-1). Nothing here reflects the state of mind of the user. In contrast, the representations of the mental aspect of state of a device are related to what users perceive, and are more controversial and potentially more subjective. The mental aspect of state is concerned with the state that the user projects upon the device when interacting with it, and the way that he/she then interprets and conceives the current state and consults expectations about the possible next states. It can be regarded as **state as conceived** by users.

Presumptions about the mental model of the user are an essential part of device design. As the discussion of the Digital Watch model in [BRWW01] illustrates, the user may only be familiar with one part of the mental model for the device as conceived by the designer. Effective ways of studying the mental aspect of state are significant for device design because it is generally hard to tell what is going on in the user's mind.

The mental state attached to users cannot be directly exposed. Even random interactions with a device may coincidentally appear to be sensible. In order to 'understand' the mental state of the user, we typically need to set up different scenarios for uses of the device. For instance, if we get rid of the white-on-black colour of the invalid buttons, as shown in Figure 5-5(b), we may probe the mental state of the users by observing and recording the pattern of their interactions. From this change to the device, we can assess whether the user's interaction with the Jugs device is merely based on choosing available options or is based on his/her perception of the status of the jugs from the visualisation. For instance, we would like to be able to reveal the fact that Jug A is full, but users think Fill A is a valid option. A further modification of the Jugs device might involve translating the text on the buttons into Thai and/or rearranging the order of the buttons on the display. This might indicate whether the user understands the concept 'jug A is full' and can interpret the words 'Fill A'.

The modifications of the Jugs device discussed above are easy to make (cf. Figure 5-5(b) because the device is realised as an ADM artefact. The simplicity of the redefinition required

reflects the fact that the modeller's construal of the device is captured in the scripts and agents of the artefact. Modifying the device is unlike modifying the Jugs program by editing and recompiling in that it respects the continuity of the modeller's perception (cf. Section 3.1). In particular (cf. Figure 5-5(a) and (b)), other aspects of state, such as the current level of liquid in the jugs, are unaffected.



%eden
mentalvalid1 is (bgcolor_wmenu1=="white")? "Jug A is not full": "Jug A is full";
mentalvalid2 is (bgcolor_wmenu2=="white")? "Jug B is not full": "Jug B is full";
mentalvalid5 is (bgcolor_wmenu5=="white")? "There is liquid in one jug and the jugs are not both full": …;

Mental

Figure 5-5: Changing the Jugs model to illustrate the mental state

The way in which we are representing mental state in this context is closely associated with the way in which an explorer apprehends state through experiment. With reference to Figure 2-10, the explorer is shaping the situation so that there are patterns of interaction to reflect the target mental state. In addition to shaping the situation by modifying the device, the modeller can also introduce an extra agent. For instance, by adding extra scripts to monitor the number of inappropriate interactions made by the users, we can be more confident about how well the user's mental state matches the state of the artefact. If we have confidence in such interpretation, the Jugs device may be extended to report the user's mental state before selecting the option as depicted in Figure 5-5(c) and the associated script labelled Mental.

The last aspect of state considered in this context is related to the use of the device in different scenarios or situations. When people design a device, they conceive potential situations and associated circumscribing observables within the boundary of normal use. In any actual

scenario in which the device is used there are other observables in the situation apart from circumscribing ones. For instance, imagine (Scenario 2) that the Jugs program is being used in a context where the time allowed for selecting a button is limited (cf. Figure 5-6). In this case, the observable timeout – outside the boundary of normal use of the Jugs program – is significant.

```
%eden
/* timeset is the time that the user clicks  */
proc stamptime: input { timeset = tn[1]; count=0; }
proc time: tn {count++;}
/* timeout, if the user does not interact within 1 minute*/
timeout is (count>=60)? 1: 0;
valid1 is !Afull && !timeout;
valid2 is !Bfull && !timeout;
valid3 is …                                  Situational
```

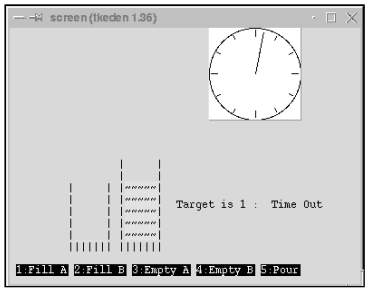


Figure 5-6: The Jugs device with the time constraint

The representation of a device by an ADM artefact enables us to extend the model into a model of the device in situated use. For instance, Figure 5-6 shows how the script for the Jugs device depicted in Figure 5-3 can be extended to represent its use in Scenario 2.

Modelling the use of a device in this way can help the designer in exploring its situated use. It makes it possible to study qualitative aspects of a user's interaction (e.g. concerning the colour of the liquid and the speed of update in the Jugs device). It can also promote creativity and suggest possible variants of the device.

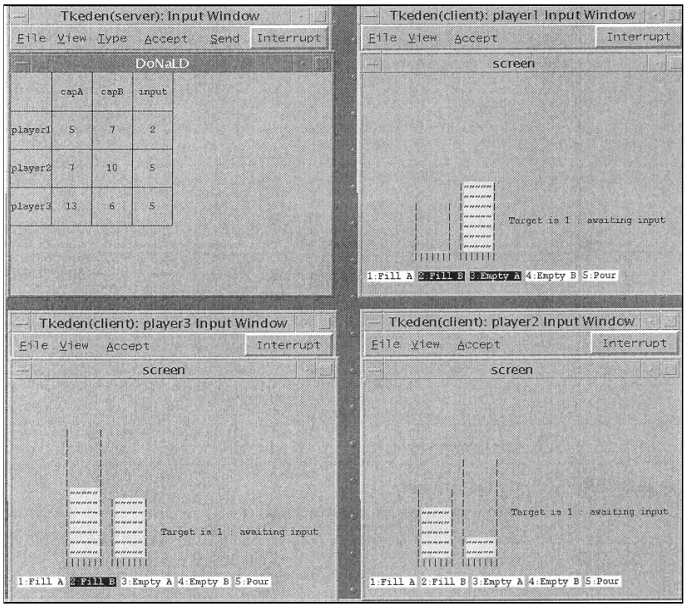


Figure 5-7: Distributed use of the Jugs device (taken from [Sun99])

By way of illustration, Figure 5-7 depicts the use of the Jugs device in a distributed environment developed by Sun [Sun99]. The observables used by the teacher in monitoring the interaction of each student with the device are included in the model.

By way of further illustration, Figure 5-8 depicts a variant of the Jugs device model in which the liquid in the jugs evaporates according to the temperature. In this situation, we have to reconsider the possible options for interacting with the device. The inclusion of Pour A and Pour B options is necessary since a state in which 'Jug A is not full' and 'Jug B is not full' at the same time, as shown in Figure 5-8(b), can now be reached. Such an extension is significantly easier than modifying the original Jugs program would be since such states are already defined in the Jugs device and are modelled independently of any particular behaviour.

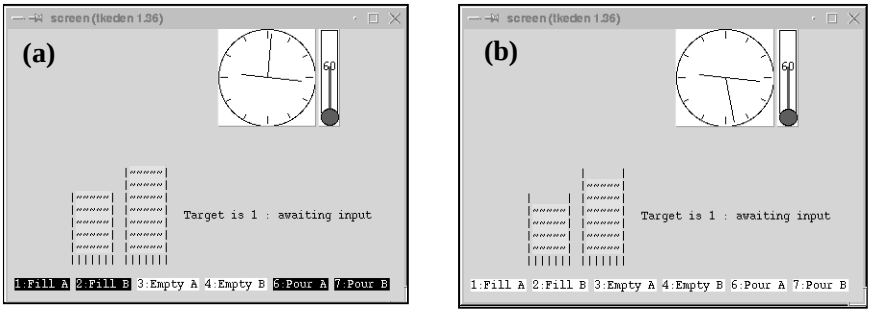


Figure 5-8: The Jugs model with the evaporating liquid

The next section revisits the two agendas proposed in Section 5.1: comparing the structure of traditional devices (as represented by classical programs) with that of ADM devices; and comparing the design and use of a traditional device (as represented by program specification and requirements analysis) with the development and use of the corresponding ADM device. These two agendas, to be addressed in Sections 5.3.1 and 5.3.2, are respectively concerned with the internal and external semantic relations of a device (cf. Figure 2-12). Section 5.3.1 focuses on the explicit and internal aspects of state and Section 5.3.2 on the situational and mental aspects of state.

5.3 ADM devices and program devices

As has been illustrated in the discussion of the Jugs device above, when modelling a device with definitive scripts, we take account of both its circumscribing observables and other observables in its operational context. The circumscribed patterns of interaction amongst the device, the user and the environment are embedded in an open-ended 'space for agent action', such as is depicted in Figure 2-17. The boundary around the use of the ADM device is virtual and is established by

the modeller's discretion. Whereas the space supports exploratory interaction, the behaviour of the device is associated with agent interaction that is managed so as to follow reliable patterns and maintain orderly observational contexts. For instance, the possible interactions of the user and the response of the Jugs device in Figure 5-3 are predictable and there is no possibility that the buttons move around the screen or that the liquid evaporates.

The computational framework for MWDS depicted in Figure 2-17 identifies the ADM device as a construal of the interactions amongst the device, the user and the environment. The ADM device captures state as experienced by the modeller and the modeller's view is significant in representing and interpreting state. Because MWDS respects the continuity of the modeller's perception (cf. Sections 3.1 and 5.2), design and use can be interleaved without interference, and the role of the modeller is more aptly characterised as 'designer-user' rather than 'designer-exclusively-or-user' (cf. [BRWW01]).

In an ADM device, the state is represented by a network of observables in a comprehensible fashion. In construing the behaviour of a device, the modeller needs to explore its relation to the user and the environment by experiment. This experimental activity can be imitated by interacting with the ADM device. Relevant observables can be extracted as sub-networks and probed to determine their values and inter-relationship.

The dependency in an ADM device is well suited to representing the intimate relations amongst the SEMI aspects of state depicted in Figure 5-1. For instance, consider the scenario in which the device is a digital watch and the situation is that the watch is one minute slow. When the user resets the time as recorded internally by the watch, this simultaneously affects the display of the watch and the user's view of the status of the watch. Conceptually, one action indivisibly affects the internal state, the explicit state and the mental state. In this way, the ADM device supplies a holistic view.

5.3.1 Explicit & internal aspects of state in ADM and program devices

Internal and explicit aspects of state of a device respectively relate to the internal representation and functionality, and the external appearance and interface. The discussion of these two aspects of state is prior to that of mental and situational aspects because, when interacting with a device, we are first concerned with exploring the device in isolation. This involves understanding the internal semantic relation (cf. Figure 2-12(a)).

The internal representation of a program device is closely connected with the choices of programming paradigm; each paradigm has its own distinct characteristics for framing the solution to a problem. There are broadly two kinds of programming paradigm: those that are concerned with modelling the problem situation and those that focus on explicitly specifying solutions. For instance, logic programming and object-oriented programming involve modelling the problem situation by using logical predicates and object abstractions. In contrast, procedural programming specifies a problem solving process in terms of a sequence of procedural actions. In this thesis, MWDS has previously been compared with other paradigms for domain modelling (cf. the discussions relating to Figures 1-5, 1-6 and 5-2 and the Digital Watch model in Chapter 3). From this point onwards, we shall focus on the relationship between MWDS and programming in its narrow sense.

It is characteristic of a device that it does not require a rich model of its operational context. It is only designed to serve a particular purpose and only needs to take account of its circumscribing observables within the boundary of normal use. This is parallel to the way that a procedural program is developed from a program specification (such as flowchart or pseudo code) that addresses its pre-identified requirements. The issues of efficiency and optimisation are prominent concerns in this kind of programming. Devices are often built from pre-existing components; this is similar to the way procedural programs are constructed using standard algorithms that are optimised to deliver preconceived functionality.

We can interpret an ADM device as a prototype for a program device. It expresses the relationships between abstract circumscribing observables that have to be maintained in normal use. For instance – when it is viewed as a device – the original Jugs program has to maintain the relationships between observables that represent the internal and explicit aspects of state depicted in Figure 5-3. Note that the ADM device is a prototype rather than a conventional input-output specification; experiential issues (such as the speed of the screen update and how the state is apprehended by the user) are significant.

In this section, we shall focus on comparing ADM devices with *procedural* program devices. Procedural program devices have the essential characteristics of the classical von Neumann computer as an artefact. Such program devices only support very primitive kinds of observation and action. These reflect the nature of the stimulus-response behaviour associated with machine code primitives, such as retrieving data from and storing data to locations, decoding the operation code of an instruction, fetching the next instruction or branching. Data

structure provides a means to organise data and manage interaction with data so that – subject to treating complex values as atomic – the effect of richer observation can be reduced to such primitive observation. It allows observation and action that involves evaluation and assignment of complex aggregates of primitive values. Both observation and action follow a fixed recipe prescribed by the data type of the variables being read or assigned.

The execution of a program device can be conceived in terms of the model for agent action in Figure 2-17. There is one agent, viz. the interpreter of the machine instructions. The observables for this agent are the sequence of machine instructions, the program counter, data locations, registers and their contents. The next action of the agent is determined by the program counter. The organisation of the instructions and data is closely prescribed since the interpretation of the instructions proceeds sequentially and essential constraints need to be met before an instruction is meaningful. The data representation in the program device does not adequately reflect the richness of human observation. Consider, for example, how a quadtree data structure is used to represent graphics, or sorted list is used to reduce the comparison of real-world entities to numerical comparison of indices. To overcome this problem, we limit the interaction with the device to well-engineered stable elements of the situation (cf. reliable paths in Figure 2-17), restrict the nature of the task to be performed by the user, and build data structures to exploit these constraints (cf. the way in which the organisation of a library enables us to access a book without needing to consult its observable characteristics in detail).

The preceding discussion of the machine as an agent is only justified if the processing of data structures is physically expressed in the program device in an appropriate way. The manipulation of data structures can only be viewed as observation and action if operations on complex values are 'atomic' and is suitably embodied in the explicit state for interpretation by the user. For instance, in a typical animation, the screen should have an appropriate resolution and refresh rate, and complex data structures should be totally evaluated before updating the screen. These are implicit assumptions that are typically made by programmers when interpreting a program as a device (cf. the experientially mediated internal semantic relation in MWDS depicted in Figure 2-12(a)). The Model-View-Controller (MVC) architecture [Olsen98], initially designed to support the multi-windowed interactive Smalltalk-80 interface, is an informal framework for managing the relationship between explicit and internal aspects of state that is particularly well-suited to the design of a program device.

The display and interfaces of a program device – the explicit aspect of its state – are built on top of its internal state. They provide a visual presentation of internal state and interfaces for the user to interact with the program device. To some extent, the relationship between internal and explicit state is expressed in a traditional program specification, which prescribes the pattern of user interactions corresponding to the requirement. The explicit state also involves an embodiment of the internal state that can only be addressed in the design and implementation phase.

Within MWDS, the data representation directly reflects observables in the external referent and the relationships amongst them (cf. Figure 1-5). This resembles the way in which a relational database can be seen as a data model of part of the real world in which the organisation is interested [Cart95]. In the relational database, the design of the tables is guided by the functional dependencies between data described in Codd's relational model [Codd70]. A relational table can be viewed as a form of data structure; it is not only meant to correspond (as a set of tuples) to a family of physical objects, but also (as a relational scheme) to the abstracted ideal of one kind of object [Kent78]. The nearest counterpart of the table in MWDS is an ADM entity without actions.

Because MWDS emphasises the representation of observables, it generally implements the internal state of the model as a side-effect of developing the explicit state. The explicit state of a model can often be considered as determined by observables in the external referent. For instance, in the Room Viewer model (see Chapter 3), the script is derived from obvious visual features such as corners, doors and walls in the room. The Scout and DoNaLD definitive notations are designed to specify the visual state of the model, and generate an internal representation that is directly linked to the explicit state of the model via dependency. The internal and explicit aspects of state are kept consistent in this way. In particular, changing the internal state will automatically affect the explicit state. This approach to implementing the explicit state can be helpful in giving the user or the modeller a comprehensible view of the current internal state. Where a visual feature has an associated interaction, the use of Scout can supply an interface to dictate the protocol by which the user can alter the state [BY90]. Consider, for instance, the mechanism for opening and closing the door in the room viewer.

MWDS can also be used effectively when developing explicit state to match given internal state. The multiple views of a model that can be conveniently specified using the MVC architecture, can also be easily implemented with a definitive approach. This is because the

internal state is represented by a definitive script and the various display representations that are based on diverse interpretations of the internal state can be connected by introducing new dependencies into the script. The principle is similar to the introduction of views in a relational database, but is generalised in this context to address different visual representations of the data (cf. Figure 2-9).

The way in which MWDS deals with observation and action provides a different perspective on the machine as an agent. MWDS induces activity at the machine level that is different in character from that described by the orthodox use of data structures. It also gives prominence to matters of indivisibility in machine execution. For instance, in the implementation of Eden, maintaining dependencies in a network of definitions takes precedence over all other kinds of machine processing. Whereas a traditional procedural program processes a data structure following a fixed recipe and separates the data representation from the program control, in MWDS, the dependencies dynamically determine the recipes by which variables are read or assigned and so connect the data representation with the program control. For this reason, the mechanism for implementing a device used in MWDS is more faithful to the way we construe observations and actions. This mechanism is oriented towards the representation of physical state rather than abstract algorithms or behaviours.

In MWDS, rich observations can be represented by combining dynamic definitions with structured data using hierarchical data dependency. For instance, in the Car History model (cf. Listing 3-3 and Figure 3-11), the possible cars are represented by the following set of definitions:

```
cars is [“Ford_Escort”, Ford_Escort, “Ford_Orion”, Ford_Orion, “Ford_Fiesta”, Ford_Fiesta, …];
Ford_Escort is [“model”, fe_model, “exhaust”, fe_exhaust, “engine”, fe_engine, …, ];
fe_model is [“Mk1”, Mk1, “Mk2”, Mk2];
```

This script can be interpreted as a dynamic data structure that represents the data stored in the model together with the dependencies between the data. Each definition is a list of data variables together with their associated identifiers. When evaluated, each definition can be regarded as a table similar to a table in a relational database system (cf. Figure 3-11), but the structure of the table is not fixed.

The difference between data representation in traditional programming and MWDS is depicted in Figure 5-9. A definitive script represents data as a collection of discrete observables dynamically linked by dependency. In a traditional program, the data is stored in a more rigidly structured fashion so that it can be referenced systematically by the program control.

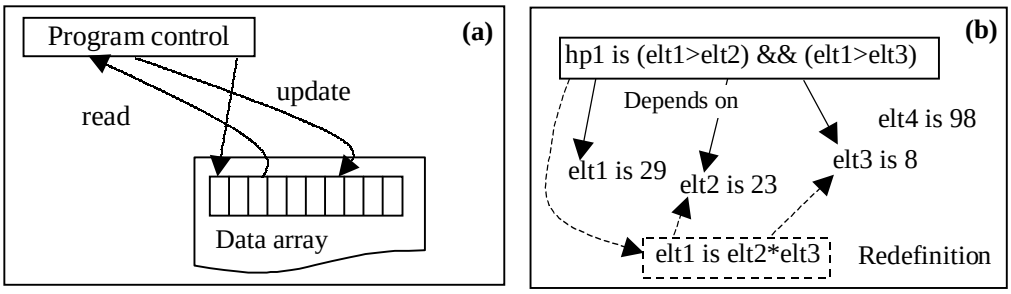


Figure 5-9: Data representation in (a) a traditional program, and (b) MWDS

Illustrating internal and explicit state in ADM and program devices

Illustrative example 1

MWDS represents internal state directly in that each variable is the counterpart of an observable. Each state of a definitive model can be interpreted and comprehended by the modeller. The internal and explicit states are typically indivisibly linked and correspond to each other state-by-state. The accessible internal states of a program device are determined by the available procedures and the predefined user control. For instance, in the Jugs device, we can set the content of jug A to the value 14 (cf. Figure 5-10(b)). In contrast, in the program device, we can only assign the content of jug A to 14 in passing (e.g. by emptying and filling jug A). In addition, if a state is not within the scope of the normal behaviour of the program device (e.g. if contentA exceeds capA), it is inaccessible.

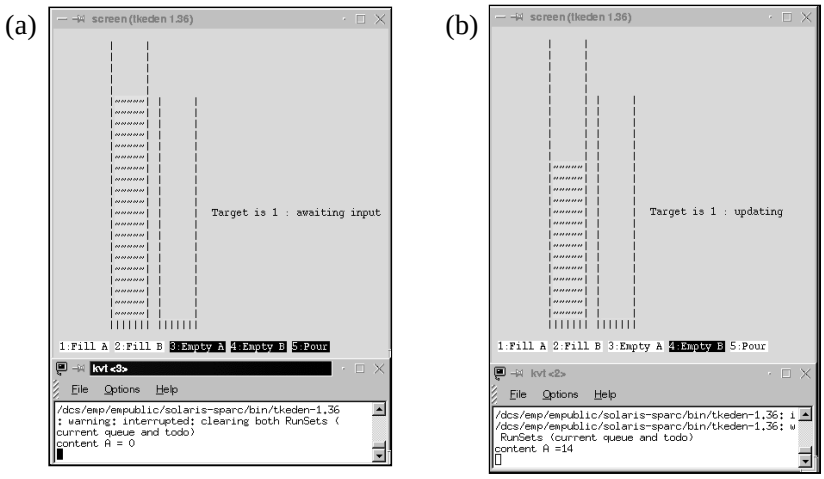


Figure 5-10: The Jugs model: (a) conventional, (b) definitive approach

We can get insight into the different ways in which an ADM device and a program device link internal and explicit states by modifying the Jugs device. Figure 5-10(a) shows a 'rogue'

variant of the Jugs device that has characteristics typical of procedural program devices. In Figure 5-10(a), a while-loop is used to update the level of the liquid in the jugs directly. When we interrupt the process of filling jug A, the explicit state does not correspond to the internal value of contentA. The objective of this procedural implementation is to save computation by reassigning of contentA only when jug A has been filled. This may increase the speed of updating the display, and does not cause problems in normal use, but the relationship between the internal and explicit states is not preserved.

In contrast, the explicit state of the Jugs device model in Figure 5-10(b) directly corresponds to the internal state of the device. Even when the device is interrupted, the internal and explicit states are consistent and correspond to the state that is obtained by directly assigning the appropriate value to contentA.

Illustrative example 2

In this section, we consider variants of the Jugs device that illustrate the significance of the internal semantic relation (cf. Figure 2-12(a)) where the functionality of the device and the capabilities of the user and the computer are concerned.

We first consider the scenario where we are not concerned with the number-theoretic problem underlying the Jugs device (cf. Figure 5-11 without the optional status string). In MWDS, we can try to represent the state of an artefact as realistically as possible. The Jugs device can be adapted to make the animation of filling, emptying and pouring the liquid more realistic. For this purpose, each unit layer of the liquid is defined by a line of pixels as depicted in Figure 5-11(a).

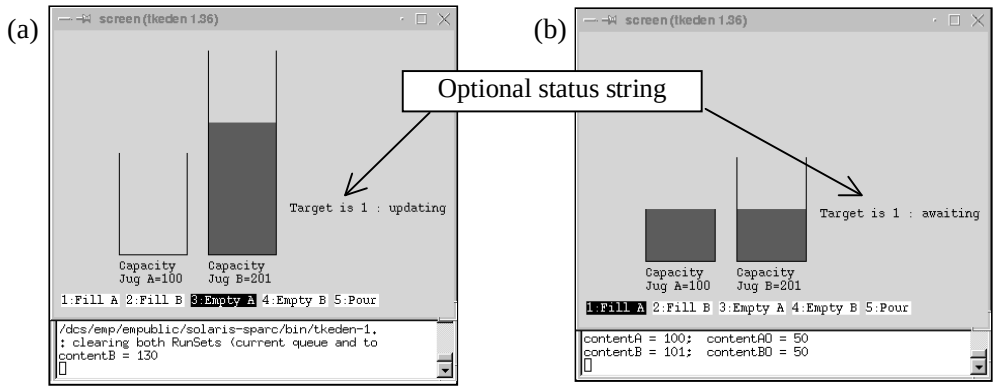
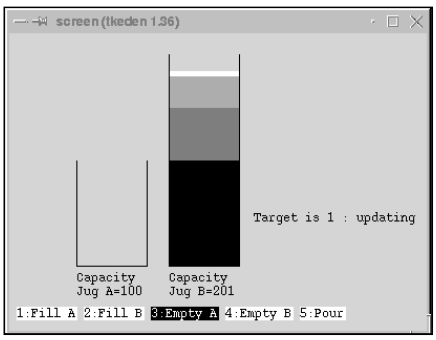


Figure 5-11: The Jugs device: (a) pixel per unit display, (b) pixel per 2 units display

In Figure 5-11(a), if the capacity of Jug B increases to 300, the computer display may not be able to cope with it. At this stage, we may have to scale down the visualisation and as a result of this we cannot have a state-by-state correspondence between the internal and explicit state any more. Figure 5-11(b) shows a variant of the Jugs device where the display of the liquid is updated whenever contentB increases by two. The visualisation is updated intermittently as a result of the redefinition of variable contentBO, defined as contentB/2. In this context, the correspondence between the explicit and internal states is one-many.

If we try to interpret Figure 5-11(a) and Figure 5-11(b) as Jugs devices (restoring the optional status string), we encounter problems. In Figure 5-11(b), we cannot distinguish visually between the situation in which contentB is 100, and the situation in which contentB is 101, when the target can be reached in a single operation. In Figure 5-11(a), the visualisation of the liquid level cannot be easily interpreted by the user as an integer. To address this problem, we can develop the richer visualisation as depicted in Figure 5-12. Note that we can easily adapt Figure 5-11(a) so that it represents jugs with capacities 5 and 7 using 20 lines of pixels per unit of liquid. A possible motivation for this modification is to make the visualisation of operations in the Jugs device more realistic. In this context the correspondence between the explicit and internal states is many-one.




```
%scout
point wconA10start={wconAend.1-widthA,
                    wconAend.2-conA10};
window wcontentA10 = {
        type: DONALD
        box [wconA10start, wconAend]
        bgcolor: "gray70"
};
window wcontentA50 = {
…
%eden
conA10 is (contentA/10)*10;
…
```

Figure 5-12: The richer visualisation for the Jugs device in Figure 5-11

Optimisation in program devices

It is expensive – even when it is feasible – to make a device that can make rich observations of its environment. For instance, it is hard to build a digital watch to reset the time by itself or to repeat the alarm on observing that the user stays in bed. For this reason, synchronising the state of the device with the state of its environment typically involves interaction between the user and the device. This means that there is an incentive to optimise cost and efficiency by minimising the

dependence on regular and close observation, and it is commonly the case that the state of a device does not reflect even its circumscribing observables very closely.

In a program device, strategies for reducing observations are typically based on assigning the values of observables rather than maintaining them by a definition. This leads to data that is cut off from further observation wherever possible. Optimisation of this nature relies on the fact that the patterns of interaction with the user are restricted and the organisation of the control and data within the device is rigid and stable. By way of illustration, in the context of Figure 5-10(a) the observable contentA is only refreshed when the visualisation shows jug A to be full. This optimisation works on the assumption that the user cannot interrupt the visualisation process and that there is no mechanism that can change the value of capA during the update.

```
[Eden definitions]

Afull is contentA==capA;
valid1 is !Afull;
valid3 is contentA != 0;
```

```
[Eden functions]

func FAfull{ Afull = (contentA==capA); };
func Fvalid1{ valid1 = FAfull(); };
func Fvalid3{ valid3 = contentA != 0; };
FAfull(); Fvalid1(); Fvalid3();
```

Listing 5-1: Converting a definitive script to a procedural program

In an ADM artefact, the scope of interaction is so broad (cf. Figure 2-17) that any 'optimisation of observation' undermines its quality as a construal. It is only when we first identify and then rigidly impose the boundaries for interaction with the artefact that we can introduce optimisation. We can regard MWDS as providing a very open-ended and flexible environment within which to model the design of a device using an ADM artefact. Observables are introduced into the model bit by bit. A model is gradually developed and it is always open to modification. However, when the designer has identified an ADM device to represent the design model, he/she can circumscribe the model by imposing the boundaries for interaction.

Once the boundaries for interaction are imposed, there are systematic ways of replacing definitions by procedural constructs. For instance, using a technique introduced by Yung [Yung96], the Eden definitions in Listing 5-1 can be replaced by the Eden functions, on the assumption that the relevant observables are not going to be redefined.

5.3.2 Situational & mental aspects of state in ADM and program devices

The previous section has discussed the representation of internal and explicit aspects of state in ADM and program devices. This section will discuss how the situational and mental aspects of

state are represented in ADM and program devices. The internal and explicit aspects of state relate to the internal semantic relation and to the design and implementation aspects of software development. The mental and situational aspects of state relate to the external semantic relation and to the requirements and specifications aspects of software development. The previous section has introduced the idea of using an ADM artefact to model the design and use of a device. We can compare this activity with the traditional approach to developing a program device.

In the traditional view of programming, the situational and mental aspects of state are studied and predefined in the early stages of model development. This view is particularly appropriate in the development of a program device, where optimisation to suit specific predetermined functions is the central concern. For instance, with reference to Figure 5-2, the model of the mental aspect of state has to be conceived before the design and implementation of the digital watch can proceed. This is consistent with what Benyon and Imaz [BI99] identify as the 'elicitation metaphor' for requirements:

> "Elicitation suggests that requirements are mental representations to be extracted from the users' heads to have a complete list of requirements."

In developing a program device, the business of asking users about their mental model is handled in the requirements analysis. Diverse situational states for the device are set up to correspond to users' requirements. Various forms of specification have been devised to describe the features and patterns of use of the device in a precise way. The purpose of a specification is to express users' requirements and to circumscribe the situations in which the device will be used. The specification of the mental state of the user is to be interpreted within a preconceived situation. For instance, in the context of Figure 5-2, we can imagine that the user is updating the UK time on the digital watch display to correspond to the current time in Tokyo. The possible situations for the use of the device are conceived in advance in terms of abstract observations (e.g. 'current time') and conventions for interaction. This may impose constraints on the actual situations that arise in practical use.

It is recognised to be difficult for the designer of the device to understand the mental and situational aspects of state from the perspective of the user. Use-case analysis [JCJO92] is an example of a software activity intended to help a designer to comprehend what users may require from a program device. This activity is concerned with a negotiation between what the user needs and what the designer can/will deliver.

The behaviour of a classical program cannot be interpreted without framing the roles of the mental and situational aspects of state in its execution. Once the program is written, the interpretation of these roles is fixed, and does not change during execution. Preconceiving these interpretations, and expressing them in a formal specification using notations with objectively defined semantics, such as Z-notation [PST96], CSP [Hoare85] and WP [Dij76] has many positive benefits where ensuring the quality and fitness for purpose of a program device is concerned. Because the specification of requirements frames the boundary of normal use for the device, it impacts on all aspects of subsequent development and use. As Spivey writes in [Spiv88]:

> "A formal specification can serve as a single, reliable reference point for those who investigate the customers' needs, those who implement programs to satisfy those needs, those who test the results, and those who write instruction manuals for the system."

The characteristics of a program device (cf. Table 5-1) are very significant in influencing the way it is developed through establishing the boundary for its use and optimising its design and implementation to take advantage of this boundary. A device developed from this perspective becomes rigid and fixed to serve a particular pre-studied situation with a pre-designed interaction pattern. It can be used to accomplish particular pre-perceived goals and it does not allow a user to interfere with the mechanism inside. The internals of the device become a mysterious black box for the user, who may learn all the relevant interactions from a written manual.

The traditional approach to requirements analysis for a device puts the emphasis on planning the user's interaction with reference to idealised observables. In the absence of a prototype, the designer relies on requirements elicitation that assumes that the user's knowledge can be articulated. In its emphasis on the significance of experience and experimentation, the use of MWDS to construct an ADM device is closer to prototyping. Conventional prototyping and simulation embodies explicit aspects of state of a device in ways that typically are not well-integrated with other aspects of state. For instance, where interaction is possible, this may be idealised in a way that only partially reflects the implicit assumptions about how this interaction is mediated by actual observables (cf. the way that interaction in a game of Noughts-and-Crosses is explored in [BJ94]). In contrast, MWDS tends to represent and handle the SEMI aspects of state in parallel. The shift from a traditional empiricist to a 'Radical Empiricist' perspective [James96] is important in this context – observables are not only derived from the primitive sensory elements of experience, but can also involve more sophisticated cognitive processes

[Bey98]. Conceptual observables, such as 'the digital watch does not keep accurate time', which are concerned with human interpretation and are not directly visible, are represented in the implicit knowledge gained through interacting with a model (cf. Figure 2-11). Once we identify such observables, we may also be able to introduce definitions to represent them explicitly.

MWDS casts the modeller in the role of an experimenter rather than a human robot. It focuses on what goes on in the experimenter's mind during his/her interaction with a prototype device. The modeller can act as a designer or a user by adopting different perspectives on the prototype. The modeller can explore his/her mind via interacting with the prototype. Interaction with the prototype and the observation of stimulus-response patterns generates the complementary knowledge about state that has to be carried in the designer's (respectively user's) head to make sense of the design (respectively normal behaviour) of the prototype device. This activity is characterised by Sun [Sun99, SRCB99] using the metaphor of 'requirements cultivation'. Making the prototype device with MWDS enables us to take a broader view of requirements. For instance, in principle, we can consider the impact that the use of devices can have upon the mind of the user, and their broader implications for users in social, cultural or administrative contexts [BRWW01].

As explained in Section 5.2 and illustrated in Figures 5-5(a) and (b), the way in which MWDS respects the continuity of the modeller's perception makes it possible to interleave design and use without interference. This close integration between device design and use in MWDS enables the user's view and user's need to be taken into account during development. Many concerns (e.g. how the designer conveys the protocol for interaction with the device to the user, how the user understands the designer's intentions, and how the designer-as-user understands and learns what is required of the device) are essentially addressed at the same time during building and interacting with the prototype device. The user's interaction with the model may be monitored so as to reveal the consequences of common mistakes and misconceptions. Practical steps that can be taken to eliminate these through redesign can be explored.

MWDS is not a methodical process but an interactive activity that involves elements of serendipity and discovery, and there is no presumption that preconceived patterns of action will achieved specified goals [Bey97]. The situational state that is associated with the normal use can evolve during the development of the prototype. An experimenter can refine the existing observables or include a new set of observables to deal with new situations as they arise. This has been illustrated in the Jugs model when the situation evolves so that the liquid evaporates, as

depicted in Figure 5-8. The 'Vending Machine for Chocolate' model (VMC) to be discussed in the next section will be used to illustrate how MWDS differs from traditional ways of specifying devices in their context.

The VMC case study

Communicating Sequential Processes (CSP) is a standard technique for formal specification of concurrent systems introduced by Hoare [Hoare85]. This section compares the CSP specification of a simple concurrent system and a representation of the system by an ADM artefact developed using MWDS.

The chocolate vending machine (VMC) is a well-known basic case study introduced by Hoare to illustrate the characteristics of CSP. Figure 5-13 depicts the VMC specification as presented by Hoare in [Hoare85] together with a diagram to represent its associated possible behaviours.

VMC = (in2p → (lgchoc → VMC | smchoc → out1p → VMC) |
in1p → (smchoc→VMC | in1p→ (lgchoc→VMC | in1p→STOP)))

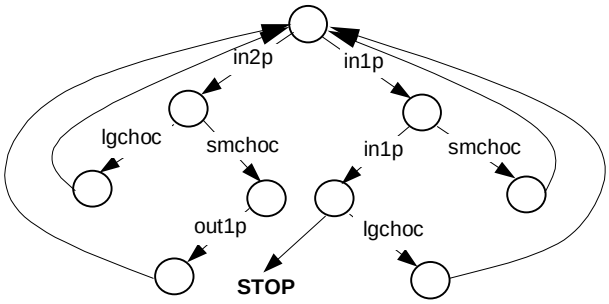


Figure 5-13: The CSP specification for the VMC from [Hoare85, p30]

CSP focuses on specifying the behaviour of a process with respect to communication and events. In the VMC process, the five possible events are indicated by *in2p*, *in1p*, *out1p*, *smchoc* and *lgchoc*. For instance, *in2p* represents 'the user inserts 2 pence into the machine'. Each potential behaviour corresponds to a sequence of events. For instance, the sequence of the events *in2p, smchoc, out1p* represents the purchase of a small chocolate using a 2 pence piece. In the specification in Figure 5-13, the possible behaviours are represented by the sequences of events or *traces* associated with the variable VMC. Each trace is determined by a sequence of choices between several currently possible events. The choice of which event will actually occur is controlled by the environment within which the process executes. The customer of the VMC decides what action to perform and the VMC behaves accordingly. The entire behaviour of a

complete system is represented by the trace of events associated with agents acting and interacting with each other and with their environment. In specifying behaviour, CSP puts the emphasis on events rather than on the agents that initiate them:

> "In choosing an alphabet, there is no need to make a distinction between events which are initiated by the object and those which are initiated by some agent outside the object. The avoidance of the concept of causality leads to considerable simplification in the theory and its application." [Hoare85, p. 24].

In modelling the chocolate vending machine using definitive scripts, our emphasis is upon the way in which agents interact rather than on abstract sequences of events. Two kinds of modelling are involved: modelling the way in which the availability of an agent action depends on the current state, and modelling the way in which the availability of an agent action is made explicitly observable in interaction. This reflects the emphasis in MWDS on state and observation respectively. The modelling activity is not primarily concerned with the abstract patterns of actions, but with how such patterns can be construed to emerge within the general framework for agent action depicted in Figure 2-17. In this context, it is essential to consider many factors that are discounted in the CSP account, such as the physical properties of the machine, how reliably the machine responds to the user's action, and the physical capabilities of the user to interpret and manipulate the state of the machine. This enables us to explore the implications of perturbing the context for the traditional VMC as specified in Figure 5-13 – in much the same spirit that a scientist might perturb parameters in an experimental context – so as to take additional factors into account. These might include matters of user discretion (e.g. what happens if the user inserts a Thai coin?), machine malfunction and agent perspective (e.g. is the user a child or blind?).

```
%eden
action1 is ["insert 2p coin", 1, !balance];
action2 is ["insert 1p coin", 2, balance<=1];
action3 is ["buy a large chocolate", 3, balance>=lgchoc];
action4 is ["buy a small chocolate", 4, balance>=smchoc];
action5 is ["get refund", 5, balance>=1];
proc vmc: action{
    switch(action){  case 1: balance=balance+coin2p; break;
                     case 2: balance=balance+coin1p; break;
                     case 3: if (balance>=lgchoc) { balance=balance-lgchoc; bank = bank+lgchoc;} break;
                     …
};
```

Listing 5-2: Extract of the script for the VMC model

Figure 5-14 depicts a definitive VMC model derived from the CSP specification shown in Figure 5-13. The model was developed incrementally through a sequence of simple modelling steps based on representing the observables that mediate the interaction between the user and the vending machine and their inter-dependency. In Figure 5-14, there are three components to the display: a text window that displays the current status of possible actions, a visual representation

of the vending machine, and the input window through which the modeller can interact in the role of VMC designer and user.

Each tuple displayed in the text window takes the form:

[*description of action, action index, validity of the action*]

Each group of five tuples represents a state of the interaction and is derived from the set of definitions in Listing 5-2 within the model by evaluation.

The variables in this extract from the script are balance, lgchoc and smchoc, which respectively represent the balance of the money for the transaction in the machine, the price of a large chocolate, and the price of a small chocolate. The script models the way in which, according to the CSP specification in Figure 5-13, the availability of the agent actions action1 through action5 depends on the current state. For instance, action3 ('buy a large chocolate') is possible provided that the balance in the machine exceeds the price of a large chocolate (balance>=lgchoc). On this basis, the actions that are valid in the three states depicted in the text window in Figure 5-14 correspond to tuples whose third component is 1. Note that the validity of actions described in the model is artificially simple, though it faithfully reflects the original CSP specification for VMC. For instance, it is impossible to insert more money if the balance for the transaction is 2 pence.

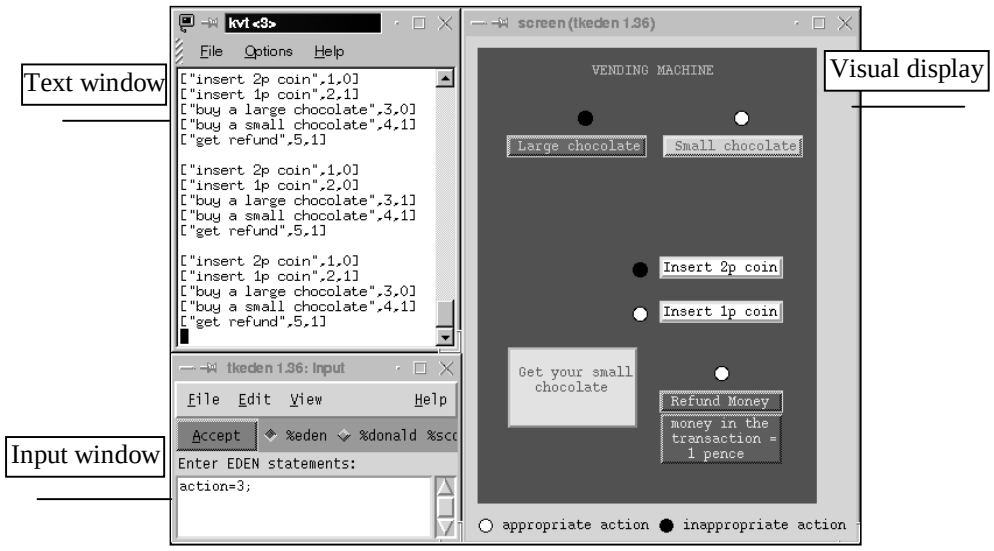


Figure 5-14: The definitive VMC model

The visual display in Figure 5-14 models the way in which the availability of agent actions is made explicitly observable in interaction. Figure 5-14 displays a scenario in which the user has inserted 2 pence, decides to get a small chocolate, then gets a 1 pence refund. In the visual

display, the black and white circle respectively represents inappropriate and appropriate actions on the part of the user. In this way, the interface declares the boundary of an ADM device. It is still possible for the user to make arbitrary interactions with the model to the input window. For instance, the redefinition `action=3` displayed in the input window will invoke an action (‘action3’) to buy a large chocolate even though this option is not available via the interface.

Modelling the chocolate vending machine as an ADM device makes it possible to take broader observations and interactions that are not necessarily associated with events and actions into account. It is not difficult to modify the actions displayed in Listing 5-2 to model more realistic behaviour. Such modification is typically quite different in character from devising a new CSP specification for the abstract behaviour. This is because an ADM device embodies a rich construal of how the chocolate vending machine operates. For instance, it is relatively straightforward to take account of how and when a situation when the machine cannot accept any more money develops. The ADM device is also typically more useful than an abstract specification in suggesting possible features to assist user interaction with the device. For instance, interaction with the model via the visual interface may disclose the need for a panel to indicate the balance of the current transaction.