4 MWDS as an activity

This chapter discusses MWDS as a modelling activity with reference to a timetabling case study. The modelling activity is viewed from two complementary perspectives:

- as developing an instrument (the 'Temposcope') to give semi-automatic support to the timetabler [BWM+00];
- as constructing an interactive situation model (ISM) [Sun99] that embodies the modeller's growing understanding and experience of the timetabling scenario.

The qualities of MWDS in respect of interaction, comprehension, discovery and extension are also described.

4.1 A case study in MWDS

In this section, we shall describe the practical use of MWDS in solving a real timetabling problem, viz. the timetabling of the third year project presentation in Computer Science at Warwick. The timetabling involves scheduling project orals for 125 students of Computer Science (CS) and Computer and Business Studies (CBS). Each oral is allocated a 40-minute timetable slot, and the timetable runs from Monday to Friday. On each day, the first available slot starts at 9 AM and the last at 5 PM so that at most 13 slots are available. During the last two academic years, the timetable has been constructed by a departmental administrator with the support of a computer-based instrument – the 'Temposcope[1]' – that has been developed using MWDS [BWM+00]. The Temposcope can simultaneously serve two complementary functions: supporting the cognitive model of the human timetabler working without computer assistance for

[1] An instrument for Timetabling with Empirical Modelling for Project Orals

matching, and providing state representations for automatic and semi-automatic timetabling activities.

Determining whether there is a feasible solution to a timetabling problem is an NP-complete problem [Gar79]. In part for this reason, practical research into timetabling has been principally directed at finding sub-optimal solutions using special-purpose techniques for constraint satisfaction. The basic strategy has been to assign penalties to undesirable conditions associated with each potential solution and to use optimisation and search techniques in combination to locate solutions with low penalty cost. This has the advantage that it can take into account both hard constraints (e.g. a member of staff cannot attend two presentations at the same time) and soft constraints (e.g. a member of staff in an external department should be assigned to consecutive slots where possible) [CRF94]. The trends in computer-supported timetabling reflect a growing concern with the broader human context; timetabling is no longer regarded solely as an abstract algorithmic problem, but as integrating data capture, data modelling, data matching, report generation and the storage of timetabling results [OptiWeb, ScheWeb].

MWDS offers an approach to satisfying timetable constraints that arguably supplies a better balance between manual and automatic activities. The aim of using the Temposcope is to support more intimate human-computer co-operation [BWM+00]. The primary objective is to construct an artefact that can embody the timetabler's knowledge of specific problem situations as they arise, rather than to automate powerful algorithmic methods. The phases in the construction of the Temposcope are broadly similar to those that accompany the development of a scientific instrument (cf. the discussion of the development of the microscope in [Hac83]). They address three principal issues: confirming that the instrument is operating as intended; ensuring that the instrument is properly situated in an appropriate environment, and becoming familiar with the most effective ways to use the instrument.

4.1.1 Constructing the data model

In MWDS, framing the problem or situation in terms of observable, dependency and agency is the essential key. Everybody has his/her own perspective on solving a problem. Some people can identify more dependencies in the solution of a problem than others. Using definitions to specify dependencies explicitly, rather than just specifying the dependencies implicitly by using procedural code, may make the model more flexible and comprehensible in use.

Dependencies are determined by our explanations for what we observe. The dependencies that are considered initially in the timetabling problem are those intrinsic to the problem, not those associated with a specific technique for solution. Timetabling is viewed as 'defined by certain observables and dependencies'. Conventional programming does not attempt to capture these observables and dependencies explicitly. Despite this, the observables and dependencies are of such primitive importance that modelling by definitions can usefully precede the conception of any methods of solution to the timetabling problem.

The basic observables in timetabling are reflected in the input data files. Examples of these are represented by the following definitions extracted from the data model:

```
1. room = ["104", "327", "313", "LL1",  "444"];
2. DA_AV = [];
3. AB_AV = [17,18,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42];
4. SBR_AV = [7,8,9,10,11,12,29,30,31,32,33,34,35,36,37,38,39,40];
5. DJKTJA_AV is union(DJK_AV, TJA_AV);
6. data1 = ["Al-Khaburi",  "Ali",  "How secure is a secure website?",  "CBS", "DA", "AB", "SBR"];
7. data2 = ["Anand", "Aradhana", "Impact & utilisation of the internet in Indian companies", "CBS",
   "YM", "MCK", "AB"];
```

Listing 4-1: Examples of definitions from the data model in the Temposcope

The data records at lines 6 and 7 specify the name of a student, the title of his/her project, their department and the initials of his/her supervisor, assessor and moderator respectively. Allocating a student to a slot in the timetable also commits his/her supervisor and assessor to attend the presentation at this time. Slots are numbered from 1 to 65 and staff unavailability is defined by a list of slots as shown in lines 2, 3 and 4. In this context, DA_AV, AB_AV and SBR_AV (in line 2, 3 and 4) respectively refer to the members of staff with initials DA, AB and SBR.

The timetabling requirements have been revised since the model was originally developed. The model was initially designed for scheduling orals in 90 time slots each lasting 30 minutes. Moderators were also expected to attend presentations. Appropriate changes to the model are relatively easy to make because of the versatility of data representation using definitive scripts. This is illustrated in line 5, where the variable DJKTJA_AV refers to the unavailability of two staff members who are jointly supervising a project, which in this case is the union of the unavailabilities of DJK_AV and TJA_AV.

MWDS is essentially a situated process. Dependencies are conceived with reference to observables and possible interactions external to the model. Thinking in a definitional way is a matter of 'education independent of programming' – we can think about concepts like observables, dependency and agency without reference to constructing computer-based models.

The definitions in Listing 4-2 are examples of dependency between observables associated with data capture.

```
1.  class is makeclass(data);                                /* a list of student name in the class */
2.  staff is makestafflist(data, [5,6,7]);                                        /* a list of staff */
3.  AVSTAFF is makeAVSTAFF(staff, avail);                       /* a list of availability of staff */
4.  avx  is map(proj2_1, [avstud], class];                    /* a list of availability of student */
5.  ASS is makeASS(class);                    /* a list of student and his supervisor and assessor */
6.  NPX is map(count, ASX);                          /* number of students allocated to each staff */
7.  ASX is map(proj1, [ASS], staff);              /* list of student names allocated to each staff */
8.   asx is map(proj2, [ASS], staff);       /* list of supervisor and assessor allocated to student */
9.   data is [data1, data2, data3, ..., data119];                       /* a record of all students */
```

Listing 4-2: Examples of definitions from the data capture in the Temposcope

In making the model, the modeller includes those observables that are significant from the perspective of somebody making a timetable. The above definitions demonstrate how such observables can be expressed by using dependencies[2]. For instance, in line 1, `class` is the collection of student names (i.e. `[Al-Khaburi Ali, Anand Aradhana, ..]`) in the timetable; it depends on `data`, so that changing the value of `data` will affect `class`, `staff` and the other definitions automatically.

The key concern addressed by the above dependencies is whether staff and students are available in the specified slots. However, in the spirit of MWDS, it is not appropriate to say that this is the only concern; the modeller always has discretion to add new observables to reflect other considerations that are not formally captured by the notion of a 'working timetable'. For instance, the distance travelled between sessions for a particular member of staff may be considered (e.g. in case of a disability). When thinking of 'someone making a timetable' from a MWDS perspective, we are not mainly concerned with procedures, but with states. In particular, the focus is typically on the timetable as a partially completed assignment of students to slots, and a provisional assignment of staff to students. It is the emphasis on states rather than procedures that allows us to take richer observables into account.

The above discussion illustrates one aspect of a general principle of MWDS: the use of definitive scripts as a data representation does not commit the modeller to specific design decisions. For instance:

---

[2] the cryptic choice of variable names in this model is due to the original developer, and is an issue to be addressed in future development.

- the model can take account of additional and provisional constraints (e.g. a CBS student may require one assessor from Computer Science and one from Business, or a member of staff may be unable to make a firm commitment to availability);
- significant design decisions can be postponed (e.g. the choice of algorithms for timetabling to be implemented (if any) and of processes for data input (if any) remains open, and can be decided at a later stage);
- new modes of observation can still be taken into account (e.g. the availability of staff member might be specified with respect to 30-minute slots rather than 40-minute slots, and the basic data model derived from external database tables).

The flexibility of the data model stems from two sources:

- the modelling activity embraces partially defined timetabling entities, and the structure of data is derived during the development of the model;
- the data model as defined (cf. Listing 4-1) can be derived by dependency from other alternative or more primitive data representations, and alternative or more complex data representations can likewise be derived by dependency from the existing data model (cf. Listings 4-1 and 4-2).

The previous discussion has addressed the role of data modelling based on dependency in the Temposcope. The next section will discuss the processes of data matching, report generation, the storage of timetabling result and the interface of the system based on using this underlying data model. This involves interpreting the state of the timetable in respect of some of the key concerns of the timetabler.

4.1.2 Developing the interface

The Temposcope aims to provide an interactive environment to help in searching for a solution to the project oral timetabling problem and to help the user to gain insight into the space of possible solutions. The user acts as a 'super agent' who can decide an appropriate slot for a student. Visualisation has an essential role here to help the user make the decision. At the current stage of development, the Temposcope has a visual interface that is limited to displaying information about the staff involved in assessing a specific student. A graphical user interface involving hundreds of Scout windows is included to facilitate the interaction of the user with the model

Several reporting functions are provided to help the user to trace the state of the timetable during his/her interaction. These are implemented via the Eden string variables MSG and msg that are maintained by dependency and record details of the current status of all the staff members and all the students respectively. Relevant functions are checking whether a staff member is double booked, a student is either not scheduled or is scheduled for more than one slot, or a staff member is unsuitably scheduled. These functions are built on the basis of the definitions of the data model defined previously. For instance, the variable d_X holds the list of staff who are double booked. The definitions in Listing 4-3 refer to observables that have a significant role in the process of constructing the interface for the Temposcope.

```
1. TT is TT_from_timeTable(timeTable);              /* the assignment of students to slots */
2. SLX is map(proj2, [join(ASS,TT)],staff);            /* the assignment of staff to slots */
3. slx is map(proj2, [TT], class);         /* the list of slots assigned to students in class */
4. d_X is combine(dblebk_X, [staff, SLX]);       /* the list of staff who are double booked */
```

Listing 4-3: Examples of definitions for developing the interface in the Temposcope

The variable timeTable records the slot and room to which each student has been allocated in the following way:

```
writeln(timeTable);
[ [ [“Al-khaburi Ali”, “104”],[“Bang Sung”, “313”], [“Barett Ian”, “007”] ], [ [] ], [ [...] ], ...]
```

In this list of lists, the first entry is the list of student-room pair allocated to slot 1 etc. This contrasts with TT, which is a list of pairs of the form [*student-name*, *slot-number*]. The presence of variables associated with different forms of representation of the same observables, and the use of definitions similar to that at line 1 in Listing 4-3 to transform one representation into another, is a common feature of MWDS. This feature reflects the rich observations of the referent that can be involved in the definitive model. For instance, the data representation in the variable timeTable directly reflects the way in which timetabling information is entered and displayed through the interface. On entering timetabling information, the user selects a room, then a student, then a slot. In displaying this information, there is a direct correspondence between the list of slots and the geometric representation of these slots. In contrast, the variable TT is adapted for representing internal dependencies between data such as appear at lines 2 and 3 in Listing 4-3.

Figure 4-1 depicts the Temposcope in use. As depicted in Figure 4-1, when a room (in this case 327) and a particular student (in this case is Adrian Clarke) is selected, the three staff members assigned to this student are highlighted (in this case SAJ, SGM and MAR), and the slots that lie in the union of the unavailability of the supervisor and assessor (in this case SAJ and SGM) are highlighted in red.

As is shown in the terminal window in Figure 4-1, the user can consult the status of an allocation by consulting the report variables MSG and msg. For instance, as shown in Figure 4-1, SGM is double booked in slot 4.

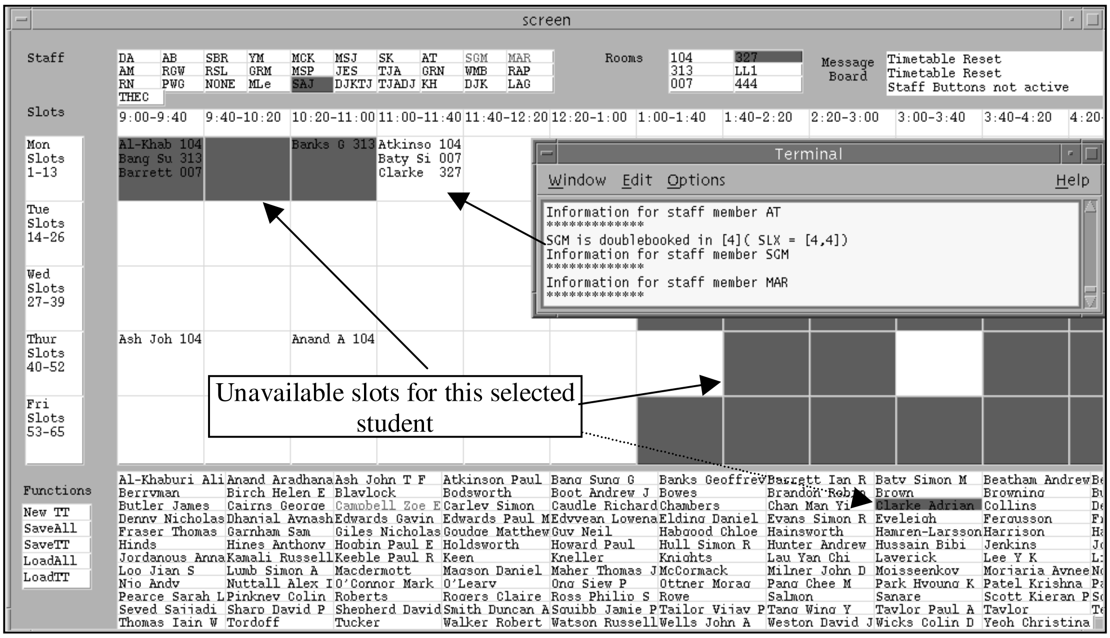

Figure 4-1: A screenshot of running the Temposcope

An essential feature of this model is the use of virtual agents to generate definitions for a large number of Scout windows in the model. Because MWDS tends to represent an entity and all its associated observables explicitly, the reproducing of similar scripts is often unavoidable. The basic mechanism for using a virtual agent named VAname involves entering definitions in a new context in dtkeden that is framed by special commands as follows:

>>VAname

enter definitions here

>>

The effect of these commands is to generate a set of definitions that is derived from the definitions entered by prefixing all variables that occur in them by the string 'VAname_'. An additional feature, illustrated in Figure 4-2, involves the indirect specification of a virtual agent name by reference to the value of a string variable (cf. temp2 and LISTBUTTON_NAME in Figure 4-2).

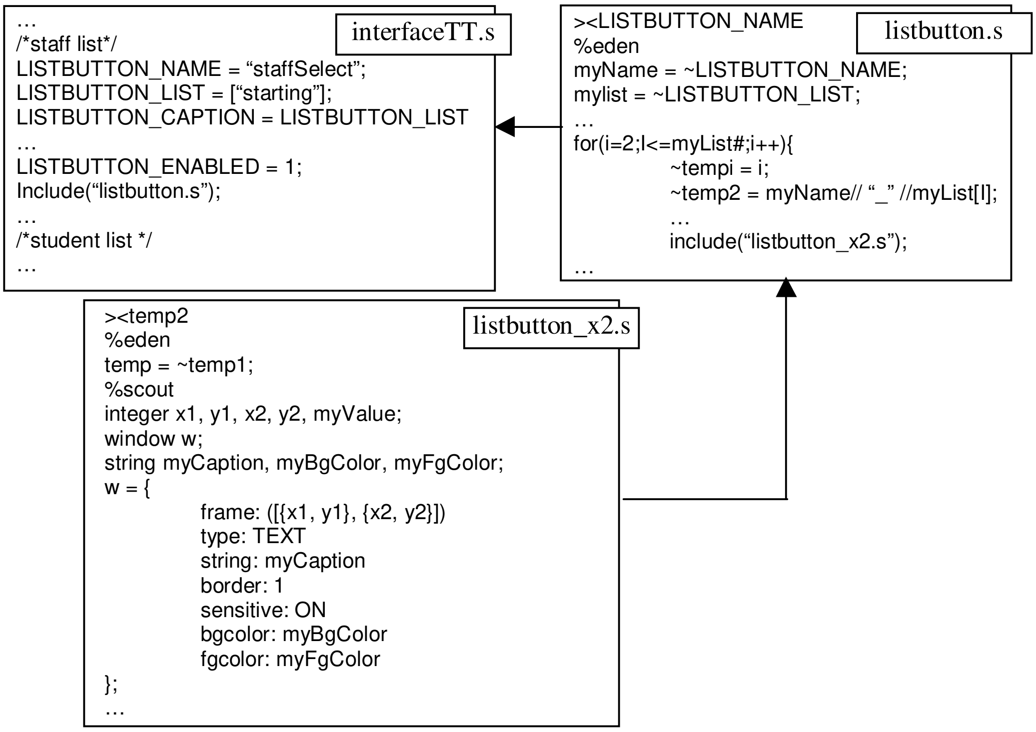

Figure 4-2: Illustrating automatic script generating with virtual agents

```
…
window staffSelect_DA_w = {
    frame: [ {staffSelect_DA_x1, staffSelect_DA_y1}, {staffSelect_DA_x2, staffSelect_DA_y2}])
    type: TEXT
    string: staffSelect_DA_myCaption
    border: 1
    sensitive: ON
    bgcolor: staffSelect_DA_myBgColor
    fgcolor: staffSelct_DA_myFgColor
};
…
```

Listing 4-4: Script extract generated by the technique described in Figure 4-2

In the Temposcope, virtual agents are used to generate all the principal components of the display interface, such the bank of student buttons, the bank of staff buttons and the array of timetable slots. For this purpose, virtual agents are used in a nested manner, as illustrated in Figure 4-2, which displays the definitions used to generate the bank of staff buttons in Figure 4-1. A generic definition is first formulated and stored in the file listbutton_x2.s, this file is then included in the file listbutton.s in different virtual agent contexts associated with the names of staff members, and the content of this file is in turn interpreted in the context of a virtual agent named ‘staffSelect’. As a result, a similar set of definitions and actions is generated for each window (cf. Listing 4-4).

The process of generating scripts with a virtual agent described above circumvents the tedious job of manually reproducing a set of similar definitions. However, it may be difficult for the modeller to comprehend the script from its specification in Figure 4-2 without executing the model. This is because definitions are generated on-the-fly and interpreted as if they were explicitly entered by the modeller. In particular, the definitions generated from Figure 4-2 are explicitly stored and maintained in the executing model.

With reference to a discussion of instruments immediately prior to Section 4.1.1, the model building activity described so far corresponds to the design and construction of an instrument. The first phase of building a text-based model to represent observables and dependencies is analogous to the preliminary experimental work of the engineer in identifying the principles on which the instrument will be based. The introduction of primitive visualisation and interface mechanisms to enable the modeller to access the data model is analogous to the construction of primitive components from which the prototype instrument can be assembled. The refinement of the user interface and rehearsal of the use of the model is analogous to the creation of the first complete prototype. At its current stage of development, the Temposcope has the characteristics of a useful instrument whose qualities have been appreciated by the departmental administrator. However, in its current form, it is far from being a finished instrument. For instance, the reporting variables MSG and msg still resemble the primitive forms of interaction that are characteristic of an instrument at an early stage of its development.

4.1.3 Using the instrument

MWDS supports activities that are more general than programming. Programming is instructing a computer to carry out a recipe. By comparison, MWDS promotes the concept of 'the computer as instrument' proposed in [BCH+01]. The use of an instrument typically involves a close and continuous engagement between the user and the computer. Ideally, the user's interaction with the computer should have an experimental quality that gives immediate feedback, so that the user can assess whether he/she is heading in the right direction. Such an activity has a form of domain or problem analysis as an essential ingredient. The way in which dependencies, functional abstractions, actions and visualisation are exploited should ideally be dictated in an interactive way by the results of this analysis.

The Temposcope provides an interactive environment for the modeller to create a model that can be shaped to suit the specific timetabling context and for the user to explore and exploit the model in parallel. Whereas conventional programming typically aims to implement the solution to a well-understood problem, MWDS typically involves finding the solution and understanding the problem at the same time. The Temposcope itself does not find a solution to the timetabling problem so much as assist the timetabler to explore and observe factors that influence the likely quality and disposition of solutions.

The concept of 'place-and-seek' proposed by Paechter [PLCP94] may be simulated manually by the user in order to solve the timetable problem. The user can select a plausible slot for a student; check whether it is appropriate and, if not, try another plausible slot. This process can be observed in Figure 4-3, where the user is deciding an appropriate slot for a student (in this case Sam Garnham). The shaded slots indicate those that are not possible for the student because the supervisor or the assessor have declared themselves unavailable. The user can choose an un-shaded slot to which no student is yet allocated and this choice will meet the hard constraints. If students have already been allocated to an un-shaded slot, the user has to check that neither the supervisor nor the assessor has been assigned as a supervisor or assessor to another student timetabled in that slot. In Figure 4-3, it can be seen that, at present, the staff member MCK is inappropriately allocated at slot number 27 and 54. The user has to reallocate students in these slots to eliminate the clashes for MCK from the timetable. The availability of staff may be changed and redefined on-the-fly. The model automatically updates the state of the timetable and will report the effect at the request of the user. The user can then try to reschedule the orals within the relevant slots appropriately.

The Temposcope model is efficient in terms of allowing redefinitions, maintaining dependencies between observables and giving a greater level of user interaction with the system. Whereas a conventional approach only gives limited information about the reasons for scheduling decisions, and typically aims to produce the schedule automatically without the user's intervention, the Temposcope user has open-ended access to the internal state of the model when making and evaluating decisions about allocating orals to slots. The user's access to the internal state of the model also means that the model can be readily adapted to suit different perspectives and meet different needs as they arise. For instance, when the user is about to schedule a student to a slot to which some students are already allocated, the user needs to determine all staff members who, as supervisors and assessors, are already assigned to that slot: this information can be derived from the observables ASS (cf. Listing 4-2) and TT (cf. Listing 4-3) by a dependency that can be framed as a new definition. The experience gained from interacting with, and extending, the model in this kind of way can lead to the discovery of new systematic techniques for finding solutions.

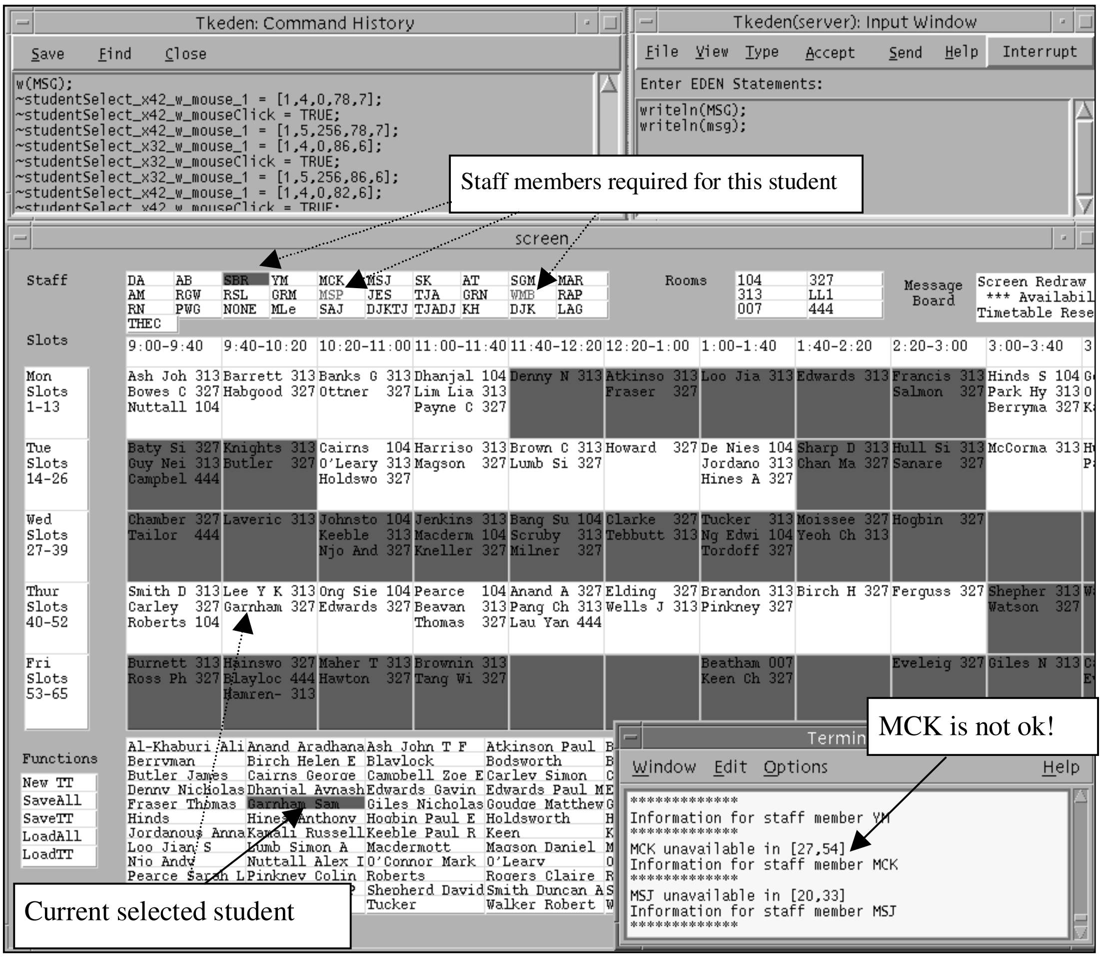

Figure 4-3: The Temposcope – an interactive timetabling model

Methods of solving the timetabling problem correspond to systematic ways of navigating around the space of partial timetables. Any computer algorithm that constructs timetables incrementally is interpretable in terms of navigating the space. Many computer algorithms might use clues or systems that the human timetabler would find inconvenient (e.g. it is not necessarily appropriate to enter students in alphabetic order). In general, the aim of MWDS is to allow a computer and a person to co-operate in the navigation towards a successful timetable. In this context, there may be observables appropriate to the timetable which give useful heuristics about how to make progress in timetable construction. For instance, a good strategy for timetabling might be to count the number of possible allocations currently available for each student, and subsequently choose to schedule the oral of a most tightly constrained student. The timetabler would not ordinarily be able to observe the identities of such students directly, but this could be made available via the definition of an appropriate observable. This illustrates how the Temposcope, as an interactive machine that maintains some essential constraints or facts about the final year project oral timetable, can offer a framework within which to develop automatic and semi-automatic timetabling algorithms.

4.2 Characteristic features of MWDS

MWDS emphasises the direct representation of the state in a situation by an artefact rather than symbolic representation. Because of the close correspondence between observables in the artefact and observables in the referent, MWDS supports interactive and experimental activities. The term 'interactive situation model' (ISM) was introduced by Sun (cf. [Sun99], [BCSW99], [BCRS99]) to describe an artefact developed in this way. The model of the timetable that is generated by using the Temposcope, as discussed above, can be alternatively viewed as an ISM for project oral timetabling.

Various modelling activities, such as interaction, comprehension, discovery and extension are essentially involved in constructing an ISM. All these activities both shape and are shaped by the external semantic relation and combine interpreting and understanding the ISM script with modelling activity that is observation-oriented and experimentally-based.

4.2.1 Interaction

Interactivity is one of the most significant features of MWDS, since the user, in the role of a super agent, can arbitrarily change the state of the model on-the-fly. Redefining a script can correspond to changing the state of the model and subsequently to changing the modeller's personal perception of the model. The modeller typically interacts with the model in order to explore its features, comprehend its internal and external representation, refine and modify it to reflect the evolving referent and experiment with it to meet subjective criteria. The way in which the semantics of definitive models is developed (cf. Figures 1-5, 2-10 and 2-11) means that MWDS can manifest particular qualities in interaction as described below:

- It supports exploratory experiment (similar to the 'what-if' feature in spreadsheets) that is useful when we need to comprehend and investigate the internal semantic relation associated with a script or confirm expectations about patterns of change.
- The acceptability of the model can be determined by the situation in mind. The content of the script is determined by the real-world situation.

- Subjective judgements are supported, as when (e.g.) in geometric design or interface design, we find a satisfactory shape or appearance by 'what looks good', or the timetabler rearranges the allocation of students to slots until he/she is satisfied.

MWDS also offers a very open environment, so that the user can interact freely with the model or make a change to a single detail of the script. There is no right or wrong way to interact with the model. Every experience or interaction counts in MWDS. In addition, the user can always restore the previous state of the model. Some definitions may not be useful or oriented to the target but they may be meaningful in relation to gaining experience and interpreting the results of experiment.

In MWDS, the referent can evolve all the time. It may be affected by changes in environments, situations and personal experiences and perspectives. An example of a real-world state that evolves and changes most of the time is a database application, since data needs to be extended and refined with the passage of time. A definitive script has open-ended characteristics to be redefined or modified without circumscription. In an unsettled world, we do not want a model that is fixed and hard to change. In Cooper's view, the inflexibility of many programs stems from the significant difference between the cognitive model of a user and that of the builder of a program [Coop99]. MWDS offers a way to construct sophisticated computer-based artefacts whose behaviour can – in principle – be customised, reprogrammed and reinterpreted by the user, and whose responses can be adapted to the user and the situation.

MWDS aims to provide an experimental environment in which the developer and the user can interact broadly with a model that is flexible to change. For instance, simple changes to the definitions in the Timetable model can lead to changes in the model behaviour, as is illustrated in the following scenarios:

- Initially, the duration of each slot is 30 minutes, but this is then changed to 40 minutes. A function convert18unto13avail that converts a binary sequence of length 90 representing availability over 30 minutes slots from Monday to Friday into a similar binary sequence of length 65 for 40 minute slots can be implemented. The Timetable model is adapted to its new context by substituting convert18unto13avail(DA_AV) for DA_AV in the definition of avail, which is the key observable from which information about staff availability is derived.

- The staff members that are assigned to each student can be changed by redefining observables such as data1, data2, data3 etc. For instance, the supervisor for the student Sam Garnham can be changed from SBR to MCK by the following redefinition:

  data[ixstud("Sam Garnham")][5] = "MCK"

  where data is as in Listing 4-2, ixstud() returns the index of a named student in the list data, and the 5th element of this list represents his supervisor. This redefinition may lead to conflicts with previous allocations. For instance, MCK may be double booked. The user may need to consult the report variables and try to correct the unsatisfactory slots.

- In the Temposcope, the staff availability can be changed dynamically by redefining a particular variable. This is illustrated in Figures 4-4(a) and 4-4(b), where redefining the availability of staff member WMB automatically updates the visual display of possible available slots for the student Sam Garnham, for whom WMB is the assessor. As shown in Figure 4-4, after this redefinition there are more possible available (i.e. non-shaded) slots for this student. This can be regarded as a 'what-if' exploratory experiment.

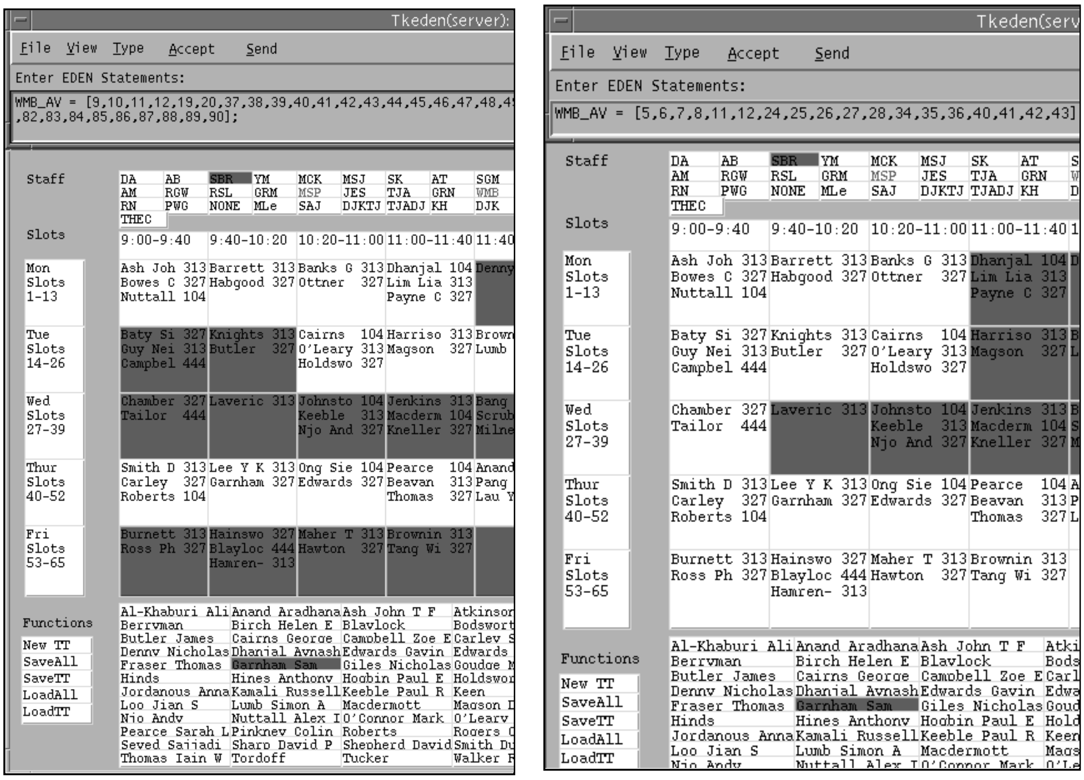

Figure 4-4: (a), (b) Changing the availability of the staff member

Extensive interaction with an ISM provides the platform for deeper activities, such as comprehension, discovery and extension.

4.2.2 Comprehension

MWDS using (d)tkeden provides features, such the query commands, the access to the history and to scripts as classified by their underlying definitive notations, and command line editing and retrieval (cf. Section 2.1.2), which facilitate the user's interaction with the model. The dependency maintenance system offers a run-time stimulus-response feature. These features help the user to make redefinitions and to understand their effects.

By changing definitions in the model and observing the consequences, the user can comprehend the behaviour of the model. Experiment can be a way to discover or confirm the interpretation of a definition. For instance, changing the availability of the staff (as shown in Figure 4-4) will affect the possible allocated slots for a student who has SBR as a supervisor or assessor. By observing the change and querying the availability of these two staff members, we can learn that the shaded slots are determined by the union of unavailability of these two staff members. A change in the availability of staff in a real-world situation often affects the state of the timetable being built. When an ISM accurately reflects behaviour in the real world situation, the user may be better able to understand the real-world phenomenon through interacting with the model.

Definitive scripts can support a large variety of ways of designating or identifying observables. This is of interest in connection with Kent's observation about designing and evaluating a computer-based data model [Kent78]:

> "In order to design or evaluate the naming facilities of an information system, it helps to be aware of the variety of ways in which we designate things."

By way of illustration, in the context of the definitive Room Viewer model depicted in Figures 2-6 and 3-7, there are a number of variables that reference the door of the room in different ways. They include: the DoNaLD openshape door, which contains the features such as the hinge and the lock, the DoNaLD line door/door, which could be viewed as representing the door in isolation from the normal context supplied by its hinge and lock, and the 'equivalent' Eden variables _door and _door_door, which are respectively associated with the abstract structure of the door and the scalar attributes of the door (such as its dimensions and position) detached from their visualisations. In MWDS, dependency is the feature that gives integrity to such diverse representations of a 'single' object.

The situated and interactive nature of an ISM provides modes of reference complementary to 'naming and designating things'. We may be able to identify the DoNaLD line door/door as it appears on the display as representing a real door because of its location relative to the other features of the real room to which it refers. Alternatively, we may be able to recognise the DoNaLD line door/door as representing a door because of the way that it is observed to behave in 'normal' interaction with the model. In this interpretation, both the character of the interaction and the context for the interaction are significant. For instance, in recognising the line as a door, it is helpful if its motion is 'continuous' rather than discrete (cf. the introduction of the DoNaLD shape rotdoor in Section 3.2.1). It is also significant that the other features in the DoNaLD line drawing in Figure 3-7 can be interpreted as items of furniture etc. Other issues of designation also become relevant as a side-effect of the interactive nature of the Room Viewer model in Figure 3-7. The line drawings in Figures 2-6(a) and (b) are interpreted as referring to a door rather than a switch since the script includes the boolean variable open rather than on.

The above discussion illustrates the richness of the concept of 'observable' as it applies to MWDS. This richness stems from the complex patterns of inter-dependency that relate the observables associated with the definitive variables openshape door, line door/door, _door, _door_door, open as discussed above. It also helps to account for the fact that, whereas in a traditional procedural program it is often difficult to connect variables with external observables, each definitive variable in a script can plausibly represent an observable in the referent. Designating a sensible name for a definitive variable in such a way that it corresponds closely to an observable in the referent can help in understanding the role and significance of the variable in the script. The experiential ingredients that support the interpretation of variables are not simply concerned with activities such as observing interaction with line drawings, but with knowledge of symbols and real-word domains. For instance, by reading the observable WMB_AV, as opposed to XYZ_AV, we may directly reckon that this observable represents the availability of a staff member with initials WMB. We can further confirm the significance of this observable in the model by querying for its value and dependency.

4.2.3 Discovery

A definitive script can be used to represent personal views, which can reflect unresolved issues and unpredictable circumstances. The modeller observes the external situation and represent observables with respect to their relationship that he/she perceives or conceives. The model that is constructed has an open-ended character. The modeller may start with only a vague idea of the purpose for the model. He/she has his/her own initial perspective. At the beginning of development, the shape of the final model is unpredictable. For instance, when constructing a geometric model such as the Room Viewer, it is hard to anticipate all useful geometric designs and dependencies. Through interacting with the model, the developer incrementally refines the model step-by-step and eventually is able to achieve an acceptable state for the model. The implications of model building are not circumscribed and not predictable. For instance, an algorithm to solve the project oral timetabling problem may or may not emerge from the development of the Timetable model.

Objects react and respond to change throughout experimentation. The object is modified in diverse directions by the developer or user. The ISM can be viewed as an experimental artefact defined by a script which the developer can redefine in order to obtain a richer understanding of the situation. Experiment leads to new discoveries. The role of an ISM resembles that of what Gooding characterises as the construals of Faraday [Good90]. In practice, people observe and interact with objects, learn their nature and make hypotheses. Experiments will test these hypotheses.

4.2.4 Extension

Sets of scripts (such as the data model and data capture for the Timetable in Listings 4-1 and 4-2) independently represent aspects of observations. They are interrelated through definitions. To extend the model, a set of definitions, which can be totally new or just redefinitions of existing ones, can be included in the model. By adding definitions, the state of the model is changed. Similar, attaching new scripts can redefine existing scripts so that they are compatible with the new ones. The model can be extended by adding new definitions which refer to observables in the existing script. For instance, the Timetabling model has been extended so that – when it is executed in client-server mode using dtkeden as depicted in Figure 4-5 – each staff member can login, as a client, to the main Timetabling model, executing on the server, and submit their availability on line[3]. In Figure 4-5, the variable WMB_AV has been changed to reflect a change in availability. The new value of this variable, as displayed in the terminal window, is sent to the main timetable server and the timetabler at the server can decide when to update the model.

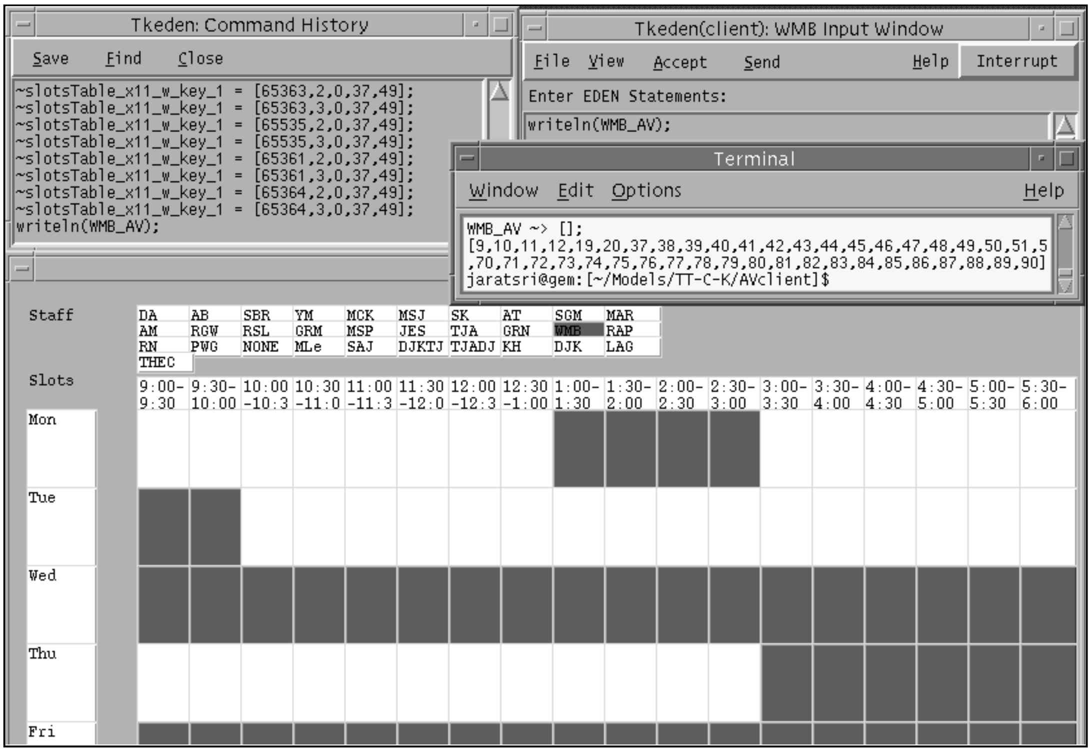

Figure 4-5: A client application for a staff member to input his or her availability

The model can be extended in terms of its visualisation modes to help the user to select suitable slots. For instance, we can write scripts to highlight the slots to which a staff member is allocated when the user clicks on a staff button. Another example of a small extension to the model attaches a workload weighting to each staff member according to the number of projects they supervise, assess and moderate as follows:

```
staffname = "WMB";
S is weight( staffname )[1];
A is weight( staffname )[2];
M is weight( staffname )[3];
Workload is 3*S + 2*A + M;

func weight{
        para name;
        auto i, s, m, a;
                s=0; m=0; a=0;
                for(i=1;i<=data#;i++){
                        if(name==data[i][5]) s=s+1;
                        if(name==data[i][6]) a=a+1;
                        if(name==data[i][7]) m=m+1;
                }
                return [s, a, m];
}
```

[3] This extension was developed by Chris Keen [Keen00] as were the web-based clients through which each staff member can update their availability via a web page.