Chapter 2
Empirical Modelling

Empirical Modelling (EM) is a novel approach to human-centred, computer-based modelling that has been developed at the University of Warwick over the last ten years. Its character and working principles embrace several different disciplines, especially psychology, cognitive science and computer science. Since the research for this thesis is based on the framework of EM, this chapter will begin by clarifying the fundamental principles and concepts of EM.

2.0 Overview

This chapter reviews and illustrates the concepts and principles of EM that form the fundamental basis for this thesis. Section 2.1 provides a general introduction to EM principles through an examination of scenarios that occur in everyday life. First, the concept of a *situated activity*, in which human agents solve problems encountered in everyday life, is introduced. A situated activity is to be distinguished from processes centred on traditional rationalistic[1] algorithms. Within a situated activity, the human

[1] The term 'rationalistic' is used in this thesis in the same way that T. Winograd and F. Flores use it in [WF86]. It denotes the view that the formulation of systematic rules can be used to capture the principles of an observed phenomenon in the real world.

factor, as the most important dimension of a soft process[2], is addressed. By comparison, traditional rationalism gives little attention to the role of human agents in dealing with diverse situations through the enaction[3] of a soft process in the real world. Moreover, the need of using the computer as a tool to facilitate the cognitive activity of a human agent is identified, in particular for a situated activity. Finally, the way in which EM research seeks to support a human-centred, computer-based approach through situated activity is highlighted.

Section 2.2 illustrates the framework of EM. Its basic concepts, observable, dependency, agent and agency, are defined in the first subsection (2.2.1). Subsection 2.2.2 then discusses the enaction of EM on the basis of constructing and maintaining the correspondences between the 'mental model' of its actor, called the modeller, the computer model, and its referent[4] in the real world. In establishing these correspondences, the modeller identifies primitive elements in the referent corresponding to the fundamental concepts above, records them by introducing appropriate definitions, functions and actions into the computer model, and also metaphorically explores, expands and experiences diverse states of the referent by interacting with the computer model. As a result, the enaction not only enhances the modeller's understanding of the referent, but also generates an interactive computer-based model as a by-product. With reference to software systems development, this by-product is exactly the evolving software system that is being constructed in the light of the modeller's current understanding of the referent. In other words, EM views a software system as a *computer-based model*, and the

[2] A *soft process* in this thesis refers to an intelligence-intensive process, such as developing a software system, designing a new car model and investigating an accident. By contrast, a *hard* process refers to a formally-defined and well-structured mechanism, such as the manufacturing process of an assembly line and the procedural instructions for operating a machine.

[3] The term 'enaction' rather than 'execution' is used in this thesis for the reason given in [STM95, p.17]: to highlight the embedding of human and computer-aided human activities in the model.

[4] The term 'referent' refers to the subject in the real world being observed by the modeller. From the perspective of EM, the referent is open and liable to change. In this thesis, the phrase "the referent associated with the subject in the real world" is often simplified as the term 'the referent'.

development of this system as *modelling*. More details will be given in the next chapter (Section 3.3).

Section 2.3 discusses technical issues supporting the enaction of EM. First, several tools previously developed for EM are reviewed (Subsection 2.3.1). By using these tools, the modeller can create a computer model and enact EM in an interactive and exploratory fashion. Amongst them, the tool tkeden has proved to be particularly successful in supporting the principles and concepts of EM. Underlying the tool tkeden is the concept of definitive programming [Yun92], explained in subsection 2.3.2. This kind of programming captures the dependencies between objects, and between objects and their properties, by declaring definitions resembling formulae in a spreadsheet. The use of definitive programming makes it possible to enact EM as a situated activity.

In the final section (2.4), the example of a hotel booking system is used to demonstrate the concepts and framework of EM. This example, developed using tkeden, also reveals the advantages of definitive programming in supporting the enaction of EM and constructing the intended software system as a computer-based model.

2.1 Meeting EM in Everyday Life

It may be difficult for a novice to understand the basic concepts and principles of EM. This is because EM is not an approach to solving a problem on the basis of traditional rationalism, where it is presumed that a good solution can be obtained by following a rigid process and abstract rules. Instead, within EM, the method of solving a particular problem is based on intelligence captured through practical experience [Bey94] – a method that human agents tacitly use to solve problems encountered in the real world.

In fact, the fundamental principles of EM are neither elusive nor intricate. Indeed, it is natural and essential for people to use these principles to solve problems in everyday

life, even though they are rarely made explicit. However, in spite of their simplicity, these principles cannot easily be formulated as rigid processes and rules. One of the best ways to understand the main principles of EM, therefore, is by considering scenarios in everyday life. As examples of scenarios:

- A person is driving through London during the rush hour and intends to arrive home as early as possible.
- Friends meet each other in the street and carry on a conversation.
- A student is using a word processor to edit a text file into a particular format.

Although such scenarios involve different situations and serve different goals, they have one important thing in common: a coherent sequence of situated actions, called a *situated activity* in this thesis, that is being constructed by the interaction between a human agent and his/her environment[5]. An action is situated if it involves conscious reference to context and choice of course of action. An action is not regarded as situated if it takes the form of a prescribed response (that is, "I am not responsible for my choice of action") or if it is an unconscious automatic response (that is, "I am not aware of my choice of action"). For example, in the first scenario, the situated activity for the driver can include overtaking other vehicles, speeding up when the traffic is good, changing to an alternative route when the traffic is too busy, and so on. In the same manner, the situated activity for one of the friends in the second scenario involves listening and replying to the speaker, changing the subject, getting distracted by other people or things, and so on.

These scenarios show that a situated activity is very different in character from an activity that is specified by a formal algorithm (such as the operation of a machine by following its instructions). Within a situated activity, each situated action, described by L.

[5] Although the term 'environment' can be used in a very broad sense, which incorporates external surroundings and the internal mind, it is used here to refer only to the external surroundings of an individual, unless otherwise indicated.

Suchman as a dynamic interaction with the actor's external environment [Suc87], is very difficult to prescribe in advance[6]. Examples can be readily found in these scenarios, such as overtaking other vehicles in the first scenario, answering a question in the second scenario and relocating a heading that appears at the bottom of a page in the third. Unpredictable events require human agents – through uniquely human capacities[7], such as intelligence, experience, knowledge and the ability to use tools – to deal with each emerging situation in ways that cannot be preconceived. For this reason, it is in general hard to prescribe a situated activity by means of a formal (or semi-formal) algorithm through which a human agent can interact with a specific environment through preconceived, fixed and well-defined methods or rules.

In fact, one of the problems with any activity formally defined by an algorithm, if it is to address the need for greater flexibility and realism, arises from its adherence to certain rigid steps or fixed methods [Gog94, Tul95]. In particular, the rationalist emphasis on regarding a specific situation as simply an instance of a more general class of similar situations abstracts an activity in the real world from its context and turns it into a routine mechanism. Because of this abstraction, and because of the inherent openness of the real world, it is hard for a formalised process to express contingent knowledge of the real world in terms of inductive inference and predetermined stimulus-response patterns [Agr95, Fey75, Suc87, WF86].

Most software process models based on a formal method require developers to follow a set of sequential activities that are well-structured and formally defined [Boe88, Boo94, STM95]. However, it has been increasingly recognised that developers in practice

[6] As Suchman argues in [Suc87, p.52], plans that are prescribed can be regarded as "resources for situated action, but do not in any strong sense determine its course".

[7] It is very difficult to find the right word to include all details of these capacities, since they are all intertwined and interdependent. For the sake of convenience, the term 'knowledge' will be used to exemplify these capacities in this thesis. However, this is not intended to suggest that knowledge is the only capacity of human agents.

have difficulty in respecting such rigid protocols [Fit96, Leh98, Rac97, SAGSZ97, Suc87, Tul95, WF86], since the real environment confronting them is usually intricate, chaotic and unpredictable. The real activities enacted by developers are to a large extent a form of situated activity. That is to say, they take situated actions without reference to a specific algorithm in order to cope with each emerging situation. From this perspective, the concept of situated activity is arguably necessary in supporting SSD (in fact, it forms the basis for the amethodical approach to be proposed by the author in this thesis (see Chapter 1)). More detailed discussion of this issue is provided in the next chapter.

A situated activity is more versatile than a formalised activity for solving problems in the real world. The most significant reason is because it is centred on human agents rather than on strict laws, algorithms or so called 'plans' arising from a particular account of the world. In effect, most plans are simply used by human agents as a resource rather than as a source of control in everyday life [Suc87]. In a situated activity, it is most appropriate to give human agents autonomy for problem-solving purposes. By reflecting on the surrounding resources, such as known information, individual experience and knowledge, and the current state of the environment, human agents can conduct reasoning in their minds to 'preview' possible results, and can consequently undertake corresponding action towards a new expected or unexpected situation. Each action undertaken, by promptly and tacitly affecting both the internal mind and the external environment, leads to a new situation and concurrently enables the situated activity to progress. In other words, situated activity, instead of prescribing preconceived activities and specifying the stimuli-response relations between human agents and their environment, highlights the importance of human agents coping with diverse situations in the real world by taking the context into account.

In short, typical problem-solving in a situated activity, as described here, reveals two features:

- The solution to a problem is *context-dependent*: it cannot be separated from the problem's context and then specified in a rigid way that does not take its situatedness into account.
- The solution is *human-centred*: human agents, whose capacities can still not be circumscribed or predefined through any formal logic or rules, play an essential role in providing a situated solution.

Certainly, a formalised process can enjoy the benefit of high quality assurance associated with an engineering discipline. However, the enaction of a situated activity that is context-dependent and human-centred is arguably necessary in dealing with real world complexity and uncertainty. Hence, a soft process is most appropriately enacted as a situated activity, that is to say, taking full account of human agents and the context.

Relying upon situated action definitely has its disadvantages. For one thing, by virtue of being human, a human agent at the centre of situated activity is inevitably error-prone and forgetful, learns slowly from experience, and can be seriously distracted by his/her environment [Hal89, Nor83, RB74]. These human factors are bound to influence not only the end-result but also the structure of the situated activity. In practice, these disadvantages caused by human factors also occur in most rationalistic models, but they are deemed to be too philosophical and open-ended to take into account. For this reason, most of these models leave the relationship between human agents and the enaction of situated activity open. In effect, such models take it for granted that the reasoning and thinking of human agents has the same internal coherence and consistency that would be expected of a mathematical model. However, this assumption is dubious when such models are interpreted in the real world, due to the openness of the environment and the inevitable fallibility of human agents.

The degree of insight the human agent has into his/her situation determines the quality of a situated action. This insight is expressed in terms of awareness of relevant

factors in the situation, and appreciation of the probable implications of action. In effect, most of drawbacks of situated activity stem from limitations of human agents in respect of cognitive activities [Nor83], such as understanding, thinking and reasoning. Fortunately, history shows that the effective use of tools can to a large extent assist human agents in performing these activities. For example, pencil and paper facilitates reasoning for most people [FP88], LOGO games facilitate the thinking of children [FSCSF88], and a physical model facilitates the understanding of physicians and chemists in solving a problem [RB74, diS88]. Today, it is widely believed that the computer is one of the best tools for human beings to facilitate the performance of these cognitive activities [Cro94, DS97, FP88].

However, it is exceedingly difficult to make effective use of the computer to support cognitive activities. For example, even though computer-based tools are already used for many rationalistic models, they can only provide limited help. This is because activity based on a formalised process is dominated by its algorithms independently of its context. Most tools developed on the basis of the algorithm for supporting the process cannot help but be context-independent. They are limited to dealing with the static information prescribed in advance rather than capturing the dynamic information emerging from the process itself. In other words, any information must be perceived and specified in the early stages of the process; otherwise the tools can take no account of it. This prohibits the tools themselves from coping with any unpredictable situation, a norm in the real world, and inevitably limits their advantages. Diverse CASE (Computer-Aided Software Engineering) tools exhibit this limitation. In view of the practical evidence, some researchers doubt whether these tools, based on specific algorithms, can provide sufficient support for software development in the real world [Blu93, BD93, Bub95].

From this perspective, it is important to use the computer in ways that best support the cognitive activity of human agents in response to the openness and unpredictability of

situated activity. Recognising this, EM seeks to provide an approach that enables a human agent engaging in a situated activity *to use the computer as an open-ended artefact to explore, expand and experience his/her understanding of a situation, as gauged by their ability to construe phenomena and anticipate the consequences of action.* This human-centred, computer-based approach has been promisingly applied to AI [Bey98], educational technology [Bey97], concurrent engineering [ABCY94], creative software development [Nes97], geometric design [Car98], and requirements understanding [SB98]. Ongoing research is applying this approach to decision support systems, business process modelling, program comprehension [BS98] and software reuse.

2.2 The Framework of EM

EM is associated with enacting a soft process characterised by the features of situated activity, but also drawing on the special capabilities of the computer to overcome the limitations of human cognition. This section gives more details of what EM is and how it works. First, the basic concepts of EM are identified. Then, the process of enacting EM is described, and close attention is given to two key activities involved in this process: observation and experiment.

2.2.1 The Basic Concepts of EM

Due to the openness of the real world, it is very difficult and provides little help to specify a situated activity in a preconceived form. For example, in the scenario of driving home (see Section 2.1), it is not sensible to preconceive the presence of a dog on the driver's way home or that the radio broadcasts that a world crisis is over. For this reason, it seems to be plausible that a situated activity can only be described in a situated manner, that is, situation by situation. In other words, a situated activity can only be understood by modelling the interaction between its enactor and his/her environment with reference to

the situations that pertain moment by moment rather than by appealing to an abstract conception of his/her behaviour. For this purpose, it proves useful to *construe a situation* in terms of the following concepts: observables, dependency, agency and agent.

- An **observable** is a characteristic of a subject to which an identity can be attributed.
- A **dependency** represents an empirically established relationship between observables.
- An **agent** is an instigator of change to observables and dependencies.
- An **agency** represents an attributed responsibility (or privilege) for a state change to an agent.

The above concepts are very general and broad. For example, the highway code can be regarded as accounting for car-driving in terms of observables (such as traffic signs, indicators, and traffic lights), dependencies (such as the relationship between the car's speed, and the speed-limit signposts and traffic lights), agents and agency (for example: a driver is responsible for stopping his/her car when he/she sees a traffic light on red). It is not too difficult to identify similar concepts in methods for SSD: for example, entities and relations in an entity-relation model [Che76], and objects and classes for an object model [Boo94, CY90]. However, as in the highway code, the intention behind these models is to use these concepts to specify a process that is in essence a situated activity in a preconceived form. As explained above, this is inadequate and provides limited help for the real process, which cannot be specified in advance. In contrast, EM makes effective use of these concepts in an open-ended fashion.

An observable in EM can be physical or abstract in nature, as illustrated by the following examples: the power of the newly designed engine, the position of the approaching aeroplane, the cry of the white seagull, and the time on Big Ben. In

conceiving a situation encountered in a situated activity, a family of relevant observables is implicated. In modelling situation-by-situation, the presence of observables can be intermittent rather than persistent, so that an observable can appear or disappear at any moment in response to each situation that is being construed. For instance, in the driving scenario, an observable, such as the dog, appears to the driver only whilst it is running past his/her car.

A dependency in EM is not merely a constraint upon observables, but reflects how the act of changing the value of one particular observable is perceived to change the values of other observables predictably and indivisibly. In this respect, dependencies play a significant part in construing a phenomenon [Bey98]. For example, in the driving scenario, it is found that the view in the rear mirror is determined by following traffic, and the car's acceleration depends on the position of the accelerator pedal. In a procedural interpretation, dependencies invoke hierarchical processes that can propagate the effect of redefining the state of any observable to all relevant observables directly or indirectly dependent on this redefined observable. For example, the brake lights are on when the brake pedal is depressed, and the brake pedal is depressed when the driver's foot pushes down on the pedal. Moreover, like an observable, a dependency need not be permanent but can instead be provisional. For example, on an icy road, the direction of motion of a skidding car no longer depends on the position of the steering wheel.

In EM, identifying agents and their agency is "associated with attributing state-change to what is construed as their primary source" [BeyMsc]. Beynon argues that agency is "in the mind of the external observer" and is "shaped by the explanatory prejudices and requirements of the external observer, and by the past experience of the system" [BeyMsc]. A typical question that helps to identify agents and agency is: "who are/is responsible (or who have/has privilege) for this state-change?" In the driving

scenario, the dog and the driver are agents when the responsibility (or privilege) for state changes, such as control over their movement, is attributed to them.

It should be noted that the concepts of agent and agency in EM are quite different from traditional agent models in the AI field, where an agent is generally defined as an entity and its ability to perform a preconceived behaviour is called agency. The specific entity is often granted or ascribed human-like mental states and is capable of interacting with its external environment in terms of these mental states [DBP93a, Rao94, Sho93, WJ95, BT94]. Hence, these models stress the conceptualised mechanism of an agent. In contrast, EM merely acknowledges the fact that agents come to be recognised by the modeller, and regards agency as being shaped by repeated observations, interactions and experimentation (see Section 4.3 for more details).

When these concepts are used in a situated, open-ended manner, the challenge of construing a situation is to provide for their computer support. In EM, definitive notations have a basic role in providing such support. A *definitive notation* is a simple programming notation for formulating definitions. A *definition* is a formula of the form x = f(y1, y2,…) similar in character to a formula in a spreadsheet[8] [Yun92]. The value of the variable x (dependent) is always equal to the evaluation of the user-defined function f with the current values of these variables y1, y2, … (the dependees[9]). Any change to the value of a dependee will give rise to a re-evaluation of the value of the dependent. For example, the definition 'A is B+C' indicates the dependency of A on B and C so that any change in the value of either B or C will cause a re-evaluation of A.

[8] More complicated definitions could be considered from the perspective of higher-order dependency [GYCBC96], which is being investigated by D. Gehring. This kind of definition is not taken into account in this thesis.

[9] The new word 'dependee' corresponding to 'dependent' is made up to denote that the value of a dependent variable is determined by those of its dependee variables.

The observables and dependencies associated with the current situation can be expressed by a set of definitions called a *definitive script*. The ordering of definitions in a script is unimportant. A redefinition of a variable automatically brings all its dependents in a definitive script to a new state through a propagation process that re-evaluates all its dependents. For example, in the definitive script 'X is Y+A; A is B+C', any change in the value of either C or B will cause a re-evaluation of X (more details of implementation issues are given in the next section).

2.2.2 Enacting EM

EM is a powerful form of interactive modelling. It allows the modeller to use the computer to create an artefact[10], an interactive computer-based model with something of the character of an engineering prototype. In order to enact EM, the modeller must first build up a virtual correspondence (as shown in Figure 2-1) between the computer model and its referent. In enacting EM, the modeller 'embeds' knowledge about observables, dependencies, agents and agency in the computer model. Interacting with the computer model allows the modeller's insight into the situation to be accessed.

Such a computer model of a situation is versatile – it can be used in dealing with many different subjects. The subject is typically an intelligence-intensive process (a soft process in the terminology of this thesis), such as developing a software system, understanding requirements, designing a geometric model, or investigating an accident. In using EM in SSD, it is envisaged that the same model may be used for understanding requirements of a software system and for its subsequent development.

[10] An artefact used for a situated activity can be a physical model, a graphic drawn on a piece of paper, a computer model, and so on. However, within the framework of EM, it is recommended that the artefact can be constructed as an interactive computer model in order to make the best use of the computer's advantages, as discussed in the previous section. Therefore, the terms 'artefact', 'computer model' and 'computer-based model' are used interchangeably in this thesis.

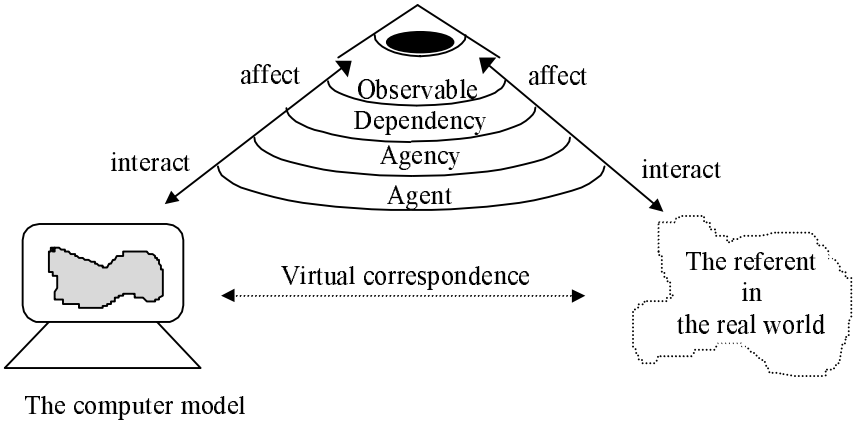

Figure 2-1. The virtual correspondence within EM

It is convenient to conceive the virtual correspondence in Figure 2-1 as established by two auxiliary correspondences. These auxiliary correspondences connect the modeller's 'mental model' of the current situation with the computer model and with the referent respectively. The concept of a mental model is introduced to acknowledge the fact that the modeller generally has insights, beliefs, and expectations of the situation (cf. the characterisation of knowledge in footnote 7) that have yet be taken into account or are in conflict with the computer model or the referent. The correspondence between the mental model and the referent is established by the interaction depicted on the right-hand side of Figure 2-1. This interaction between the modeller and the referent enables information exchange and creation. Each change that occurs in the referent, whether it is triggered by the modeller or not, may affect the mental model. The correspondence between the mental model and the computer model is established by the interaction depicted on the left-hand side of Figure 2-1. In this case, the modeller is empowered to interact with the artefact, and may also be affected by any change in the artefact. The insight gained by the modeller through establishing the virtual correspondence is

expressed in coherence between an abstract explanatory model – or *construal*[11] – in the modeller's mind, the physical embodiment of this construal in the computer model, and a situation in the referent.

In constructing the virtual correspondence, the modeller can identify primitive elements in the domain being modelled corresponding to the fundamental concepts above, and then record them by introducing appropriate definitions, functions and actions[12] into the computer model. A typical step in this process involves the identification of a dependency and the introduction of the definition into the computer model. From the modeller's perspective, a definition is "recording a dependency between the observables". From a computational perspective, the abstract semantics of introducing such a definition is typically similar to that of introducing a new formula into a spreadsheet to which a visualisation of cell values is attached. In particular, the dependencies amongst observables are automatically maintained (that is, any change in a dependee is propagated to all its dependents. More details are given in Section 2.3.2). Unlike traditional programming codes, definitions do not have to be entered and organised sequentially. Because of these properties, the construction of such computer-based artefacts is a useful vehicle for exploring and developing insight.

EM is a means of constructing knowledge in an experiential rather than a declarative fashion: the modeller's insight is expressed as coherence between expectations in the mind and the experimentation that can be performed on the computer model and/or in the referent. The principle resembles 'what if' experiments with a spreadsheet. The modeller introduces new definitions to impose a change of state upon

[11] D. Gooding introduces the term 'construal' in analysing Faraday's experimental practices. He regards a construal as "a means of interpreting unfamiliar experience and communicating one's trial interpretations" [Goo90, p.22] and argues that "a construal cannot be grasped independently of the exploratory behaviour that produces it or the ostensive practices whereby an observer tries to convey it" [Goo90, p.88].

[12] Actions are specified as procedures that are triggered by changes in the values of a particular variable. [Bey97].

the embodied construal, that is, the computer model. Almost simultaneously, the new state of this construal is mediated to the user through the visual interface, and evokes a change of state in the mind of the modeller. When this change of state is consistent with the modeller's expectations, it serves to reinforce the modeller's confidence in the way in which a situation has been construed. When the change of state confounds expectations, the modeller must determine whether the situation has been construed in an inappropriate way, for example by giving an incorrect definition, or whether a hitherto unsuspected behaviour has been identified. In the latter case, there is a creative and often surprising element of discovery that is rarely encountered in conventional modelling.

In fact, the modeller not only enriches but also to some degree embodies his/her mental model through the continuous interactions with the computer model. This is because the computer model is incrementally developed to correspond to the mental model and then to the referent. More details are provided in the next chapter.

Theoretically, the enaction of EM is unbounded since it is not possible to take all situations associated with a particular subject into account. As in everyday life, the modeller continually confronts different unpredictable situations. A new situation can cause a discrepancy between the modeller's mental model, the computer model and the referent in Figure 2-1. The modeller may interact with both the computer model and the referent in order to resolve such discrepancies. Situated interaction of this nature reflexively constitutes the situated activity of enacting EM, which cannot be prescribed by algorithms in advance. It also accounts for the openness of EM itself.

Obviously, the main crux of enacting EM lies in maintaining the virtual correspondence. With reference to the right-hand side of Figure 2-1, the modeller's interaction with the referent is not constrained by an explicit interface. Like an experimenter, the modeller may not be aware of what actions can affect the states of the referent. The interaction with the referent is open subject to empirically established

knowledge of the observables that can be changed, and the associated dependencies. Interaction with the computer model on the left-hand side must be supported in the same manner. For this purpose, the computer model must have automatic dependency maintenance, that is to say, the model must be appropriately restructured in an automatic fashion in response to any change to its elements. This feature allows the realisation of the maintenance of the virtual correspondence. The necessary supporting technique is provided in the next section.

EM does not claim that using a computer model as an artefact is the only way to model a situation. After all, the human brain, supported by paper and pen, has performed the same task quite effectively for hundreds of years. However, as explained in the previous section (2.1), the computer has unusual potential as a supporting tool for helping to overcome human cognitive limitations in information processing. Many cognitive activities of human agents, such as reasoning and remembering, can be greatly improved by externalising them to the computer. For example, in a 'what if' experiment in a spreadsheet, the modeller can 'observe' rather than 'imagine' or 'conjecture' possible results from the artefact. This helps the modeller to reason more quickly and with more confidence. In this sense, the computer model does serve as an artefact for improving human cognition [Nes97, Rus97].

However, it is evident that most cognitive activities of human agents are too complicated and sometimes insufficiently predictable to be completely automated. Recognising this fact, EM makes best use of the capacity of the computer by delegating to it routine and structured tasks that involve complex calculation and huge demands on memory. At the same time, EM highlights the role of human agents in a situated activity, allowing the modeller to carry out intuitive and situated procedures, such as the identification of observables, dependencies, agents and agency. In this respect, the enaction of EM is consistent with C. Tully's concern about the mechanism of enacting a

software process model, which on the one hand is "a symbiosis of human agent and computer" and on the other hand should be such as "not to hint at particular roles for either partner" [Tul88, p.3].

Both auxiliary correspondences are reached through various interactive activities, such as observation, experimentation, creation and so on. Since the first two are the primitive and critical activities in EM, a more detailed explanation of them is given here.

- Observation

In EM, observation, which refers to the modeller's ability to apprehend features of a particular situation directly, is vitally important. Without it, cognitive activity reverts to a traditional form: the modeller relies on imagination or conjecture without any assistance from suitable tools. Observation can be invoked on both sides of Figure 2-1, that is, to observe both the referent associated with a subject in the real world and the computer model. At least two correlated psychological events relating to the enhancement of the modeller's insight are necessarily involved: perception and connection (cf. [Hal89]).

Perception is concerned with identifying features in the computer model and/or in the referent. Connection involves associating these features with the mental model. The performance of each event is deeply bound up with factors affecting cognition, such as past experience, subjective belief, the understanding of a situation, and so on. At the same time, the result of performing both events leads to an alteration in the modeller's mental model and to the formation of a new state which influences subsequent activities. In other words, through observation, the modeller can not only construe the current situation but can also enrich his/her resources for dealing with future situations.

Perception and connection play significant roles in establishing the virtual correspondence between the computer-based model and the referent in the real world. They account for the way in which information about the referent is propagated to the

computer model via the mental model, and vice versa. By this means, the referent can be metaphorically represented by the computer-based model. For example, placing a lamp on a desk can be represented and understood as placing a circle (representing the lamp) inside a rectangle (representing the desk). In the same manner, the state of the computer model can be referred to the state of the referent. For example, changing the position of the rectangle in the computer model can be referred to moving the desk.

It may be claimed that observations are also carried out in enacting traditional process models. In a narrow sense, a kind of observation is indeed performed. However, in these models, observation is intended to pin down elements whose nature is context-dependent within a particular context. These elements, e.g. entities and relations for an entity-relation model [Che76], and objects and classes for an object model [Boo94, CY90], are preconceived, prescribed and then isolated from the proceeding process until – in view of a new functionality or context – a further change of these elements is required. In other words, this kind of observation serves to draw a line to separate the developed model from the referent. Accordingly, the developed model, which prescribes a frozen domain, becomes well suited for the use of orthodox tools and methods that are devised for implementation. This separation can make the implementation more effective and robust, but at the price of being less adaptable (details are given in Chapter 3).

- Experimentation

The choice of the epithet empirical reflects the pivotal role that experimentation plays in EM. In effect, it plays a 'creator' role for modelling a situation in EM, since it always 'creates' diverse new states with surprising discoveries that can enrich the procedure of modelling a situation. Without experimentation, modelling a situation will be reduced to 'imagining' reliable patterns of state change in the same way that behaviour in conventional programming is preconceived in response to each particular situation. In this case, the method of modelling will degenerate into what Feyerabend in his book *Against*

Method has characterised as a 'scientific' approach, which in fact is not easily capable of discovering new ideas [Fey75].

Modelling a situation is the most elusive but fundamental aspect of the EM approach. As Beynon argues [Bey98], a situation should not be "interpreted as referring to an abstract computational state, but to something resembling a 'state of mind' that derives its meaning from a relationship between a human agent and an external focus of interest and attention". Modelling a situation involves devising diverse interpretations of the relationships between this situation and its diverse state changes. To do this, an animation of knowing-by-doing through 'what if' experiments is introduced. Instead of reasoning (or imagining) possible results in his/her brain according to the current situation, the modeller changes the state of the computer model (doing) to bring about a new state of his/her mind (knowing). In this way, the modeller can enhance his/her understanding of the situation and perhaps even make surprising discoveries by exploring unfamiliar territory.

Theoretically, experimentations can be invoked both in the computer model and the referent. However, EM puts greater emphasis on the computer model. This is partly because in many cases performing an experiment in the real world is very difficult and expensive, as is illustrated by the example of developing a new air traffic control system. More importantly, the modeller can make the best use of the power of the computer as an interactive modelling medium to achieve the principled theme of EM: to explore, expand and experience the modeller's understanding associated with the subject.

It should be noted that although observation and experimentation have been discussed separately here, they are inseparably invoked by the modeller in order to maintain the correspondences between the modeller's mental world, the computer-based model, and the referent in the real world.

2.3 Technical Issues of EM

Since EM highlights the importance of empirical experience arising from repeated observation and experimentation, it fulfils the requirements for *enactability* described by Tully in [Tul88]:

> If we set out to develop models, formalisms or representations [for SSD], then there is a strong case that they should be enactable – that is, should take form of 'process programs'. Enactability simply means that human beings involved in the software process receive computer guidance and assistance in what is an extremely complex activity. Put another way, models are not just used 'off-line', as a means of studying and defining processes, but also 'on-line' while processes are being carried out, as a means of directing, controlling, monitoring and instrumenting them.

In order to support the *enactability* of EM, the technical issues of supporting the principles and concepts of EM, especially the development of computer-based tools, must be considered.

2.3.1 Tools for Supporting EM

EM aims to enable the modeller to extend, expand and experience his/her mental world through the interaction with the computer model. To build up such an interactive artefact, several tools have been developed and these will be summarised briefly.

LSD (Language for Specification & Description) [Bey86] is an open-ended notation used to account for the referent in the real world. It provides a description of "those observables that are bound to an agent (*state*), those that it is conditionally privileged to change (*handle*), and those to which it responds (*oracle*)" [BR94]. It also includes an account of the dependencies between observables perceived by the agent (*derivates*) and of the actions it is conditionally privileged to perform (*protocol*). It should be noted that this description indicates the modeller's provisional construal of subjects

and accordingly should not be viewed as a circumscribed specification, such as a requirements specification as defined in [LK95].

Within the enaction of EM, the LSD notation is useful for recording the identification and classification of agents, agency and observables associated with the modeller's observation of the referent and the model. In effect, an LSD account records the modeller's construal of state changes. Many different possible state changes and patterns of behaviour may be consistent with this construal. For this reason, an LSD account is not executable. To interpret an LSD account, the modeller needs interactive tools to realise and explore state changes consistent with the description. The tools ADM and tkeden, complementary to LSD and necessary for the enaction of EM, serve this purpose.

The tool ADM (Abstract Definitive Machine) is used to study parallel state-change, synchronisation of agent actions and openness in an LSD account [Sla90]. The modeller can manually transform an LSD account to an executable program in the ADM. An animation is then devised to give operational meaning to interaction between the LSD agents (such as what an agent can refer to in a particular state and how it can act to change the state). In this way, the modeller using the ADM can dynamically intervene and redirect the execution of this animation by interacting with this model in order to improve his/her understanding.

Another tool tkeden, one of the most successful tools for EM, is developed on the basis of the fundamental principles of EM. Its basic architecture is shown in Figure 2-2. In tkeden, there is a window-based interface based on a Tcl/Tk interpreter. This interface provides the modeller with an interactive environment to introduce new definitions into the computer model, and thus to observe their influence on the model's visualisation. Each definition is read into the Tcl/Tk interpreter as data and is stored prior to further manipulation. Visualisation of the computer model is established through two

observational tools: DoNaLD [ABH86] and Scout [Dep92]. The former is a two-dimensional line drawing tool, and the latter deals with the issues of screen layout.

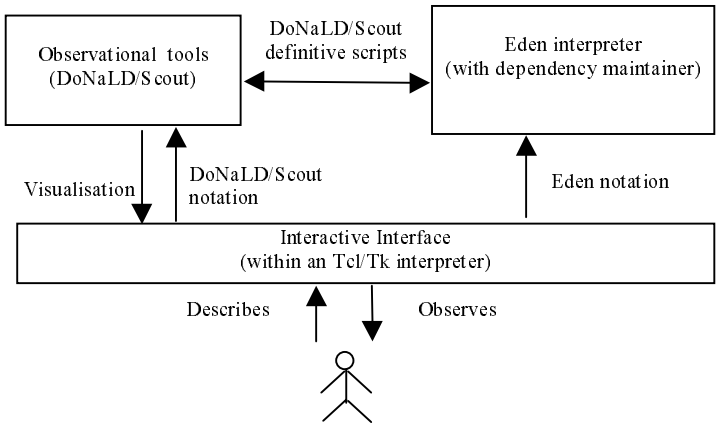

Figure 2-2: The architecture of tkeden

The core part of tkeden is an interpreter called Eden [YY88, Yun90]. This is both a definitive language for specifying definitive state transitions and also a virtual machine for maintaining the dependencies of given definitions in an interactive way. Each definition is maintained in the form of formulae resembling those in a spreadsheet. When the value of a dependee is changed, EDEN automatically propagates the change to its dependents and re-evaluates the values of these dependents. The premise that dependencies are automatically maintained is related to an indivisible state change propagated so as to reflect change in the referent rather than in a control mechanism in the programming sense. Nothing in such a model is preconceived, because no one knows what definitions will subsequently be introduced by a user.

Both tools, the ADM and tkeden, are interactive tools for supporting EM. They can be used independently and serve different purposes. The ADM focuses on the concurrent systems modelling needed in order to exhibit appropriate behaviours consistent with the LSD account. In contrast, the tool tkeden is more concerned with the visualisation of state-changes to observables and dependencies. It is often useful to

combine concurrent systems modelling with visualisation. For this purpose, translators from the ADM to tkeden (adm and adm3 – see Section 6.2.1) have been developed. These allow an LSD account to be semi-automatically translated to a tkeden model.

2.3.2 Definitive Programming

One of the most important techniques behind tkeden is definitive programming, contributed by Y.P. Yung's PhD research [Yun92]. This technique, which refers to definition-based programming, seeks to "capture the dependency information of the properties within an object and between objects by means of definitions" [Yun92, p.5]. As described earlier, a definition is a formula of the form x = f (y1, y2, …). The value of the variable x is always obtained by evaluating the formula.

Definitive programming uses definitive notations to establish a state-transition model in the computer. A definitive notation is a programming notation[13] that can be used for formulating a set of definitions. DoNaLD and Scout are two examples of definitive notations. A state of the model is then represented by a set of definitions – a definitive script – and a transition is accomplished by modifying the definitive script. This modification can involve overwriting an existing definition (redefinition) or just adding a new definition. Each such modification changes the current state of the model and leads to a new state by automatically re-evaluating the script.

To illustrate the concept of state transition in a model, consider the computer model that is constructed by EDEN by building up a definitive script step-by-step (in the syntax of EDEN and followed by its output) by introducing the following sequence of dependencies:

[13] It is described as a 'programming notation' rather than a 'programming language' because it represents only part of the information needed for general-purpose programming [Yun92, p.6].

1. Rectangle-area is Rectangle-length * Rectangle-width; writeln(Rectangle-area);

   → @[14]

2. Rectangle-length is 10; Rectangle-width is 20; writeln(Rectangle-area);

   → 200

3. Rectangle-length is 15; writeln(Rectangle-area);

   → 300

4. Cuboid-volume is Rectangle-area * Cuboid-high; Cuboid-high is 10; writeln(Cuboid-volume);

   → 3000

5. Rectangle-length is 12; writeln(Rectangle-area, ",", Cuboid-volume);

   → 240, 2400

The initial state of the computer model is established by giving a definition of observable *Rectangle-area* (step 1). By adding new definitions (step 2) and redefining an old definition (step 3), the state of the model is changed. Three different outputs, resulting from the first three steps, for the same observable *Rectangle-area* indicate that the re-evaluation is automatically executed. Given state 4, in addition to the explicit dependency between *Cuboid-volume, Rectangle-area,* and *Cuboid-high*, an implicit dependency between the observables *Cuboid-volume*, *Rectangle-length* and *Rectangle-width* is also established. This implicit dependency is demonstrated in the output of the last step in which the change to observable *Rectangle-length* (step 5) is propagated to the observable *Cuboid-volume*.

In other words, when a redefinition of an observable in a definitive script is given, a re-evaluation of the observables that are dependent on this observable is automatically

[14] In tkeden and LSD, the symbol '@' denotes 'undefined'.

invoked. The automated mechanism of maintaining the dependency between observables provides a very important basis for programming in an interactive and exploratory fashion [Yun92].

For SSD, this interactive programming technique enables the modeller to establish an incomplete computer-based model and improve the model incrementally. In this sense, programming becomes a matter of solving a problem rather than translating a specification. The translation of a specification can be accomplished in a single pass by a top-down or bottom-up approach without referring to the knowledge emerging from the on-going process. Instead, the problem-solving process as a situated activity must make progress incrementally in response to its situation and the emerging knowledge. Solving a jigsaw puzzle is a good example of the piece-by-piece solution of a task. A divide-and-conquer strategy is often used in this case. The most easily identified and more geometrically significant pieces of a jigsaw, such as the four corners, might be put in place first in order to provide further valuable information. Most solvers continue to add to the jigsaw piece by piece in response to the current state arising from the completed segments, rather than complete the jigsaw in a particular sequential order, such as from the upper-left corner to the lower-right corner, without referring to the emerging knowledge.

The value of a variable within definitive programming can be undefined and can be automatically revised. This feature enables the programmer to define variables in accord with their semantics. For example, even though its two dependees (length and width) are not defined yet, the definition of the area of a rectangle can be given (as in step 1 above). Hence, the task of programming can be accomplished by local adjustment (that is, by a piece-by-piece strategy). The incremental development feature is difficult to achieve for traditional programming, in particular for procedural programming. This is because, for traditional programming, any change to a program typically must be

performed from a global viewpoint, since it could affect other parts of this program. However, with definitive programming, each local change is propagated to the whole program and leads to the necessary re-evaluation. For example, changing the value of *Rectangle-length* (in steps 2 and 3) will cause the values of *Rectangle-area* and *Cuboid-volume* (in steps 4 and 5) to be re-evaluated.

For most traditional programming methods, an undefined variable is not allowed. For example, step 1 can cause an error of data type during compilation in most traditional programming languages [Ous98], if both variables *Rectangle-length* and *Rectangle-width* are undefined. In addition, according to the sequential algorithm provided by any of these methods, the definitions given in steps 2 and 3 do not change the value of *Rectangle-area*. In other words, dependency maintenance is not supported by these methods.

Moreover, the exploratory programming empowers the modeller to experiment with the computer model in order to enhance his/her understanding. Design is a trial-and-error learning activity [Som92, Vli93]. It is valuable to explore diverse situations in order to capture a deeper understanding of a problem and its solution through a variety of experiments. Giving a redefinition, that is, an experiment, the modeller can see – *experience* – a state change in the computer model [Bey94]. In the light of these immediately experienced state changes, the modeller can modify or qualify the virtual correspondence and, more significantly, reconstruct his/her understanding. In this respect, the theme of EM to a large extent accords with the concerns of constructivism[15]: knowing-by-doing, a very important concept widely used in education [Puf88]. The invocation of experiments on the computer model, as actions on the subject in a constructive model, is an interactive, situated mechanism whereby the understanding can be extended, expanded and experienced.

[15] Constructivism is characterised as "the continual restructuring of the relation between self and world, where world implies both palpable and ideational reality" [Puf88, p.17].

In summary, definitive programming is very helpful for the modeller seeking to enact EM in the form of situated activity. As described earlier, the openness of situated activity enables human agents to cope with varied situations by taking situated actions [Suc87]. In the same manner, it is necessary for the modeller to interact with the computer model in an open-ended manner. Moreover, in order to ease the limitations of human cognitive activities, it is helpful to use the computer as an artefact to improve understanding of a problem and its solution. With the aid of the interactive and exploratory features embedded in definitive programming, EM can serve these purposes in a significant way.

2.4 An Example illustrating EM

In order to illustrate the concepts of EM and the use of those tools mentioned in the previous section, the example of developing a hotel booking system is discussed here. To make a reservation, the customer must tell the hotel receptionist both the arrival date and the departure date. At the same time, the receptionist checks the availability of all room slots (12 rooms) in the reservation tables for those dates. If there is an available room, the receptionist puts the customer's name into the room slot for each day of the intended stay. According to the observation in the real world, two agents are identified by the modeller in response to his/her observation from the process of making a reservation:

- The customer who wants to make a reservation;
- The receptionist who is dealing with the customer's request.

Next, the modeller defines each of the agents in LSD. The LSD account of the customer and the receptionist agents could be as follows:

```
agent customer (c) {
  state  customer_giveDate (c) /* the customer c intends to make a reservation */
  oracle reservation_ok (c, d1, d2) /* the reservation for the customer c during d1 and d2 is ok */
  handle customer_giveDate (c) /* the customer c provide dates of arrival day and departure day */
protocol
      reservation_ok (c, d1, d2) == FALSE    => customer_giveDate (c) = TRUE;
      reservation_ok (c, d1, d2) == TRUE     => customer_giveDate (c) = FALSE;
}

agent receptionist {
  oracle customer_arrival_day (c, d1), customer_depart_day (c, d2), room_slot (n,d)
  handle
     customer_arrival_day (c, d1) /* customer c is expected to arrive at day d1 */
     customer_depart_day (c, d2) /* customer c is expected to depart at day d2 */
     room_slot (n, d) /* the content of room slot n at day d */
     reservation_ok (c, d1, d2)
  derivate
     customer_arrival_day (c, d1) = customer_giveDate (c) ? input(c, d1) : @
     customer_depart_day (c, d2) = customer_giveDate (c) ? input(c, d2) : @
     room_availability (n, d1, d2)
              = (∃d, d1<=d<d2, room_slot (n, d ) ! = "" ) ? "reserved" : "available"
  protocol
    customer_arrival_day (c, d1) != @ && customer_depart_day (c, d2) != @  &&
       reservation_ok (c, d1, d2) == @ && (∃n, room_availability (n, d1, d2) == "available")
         => reservation_ok (c, d1, d2) = TRUE; (room_slot (n, d) == c, ∀d, d1<=d<d2)
    customer_arrival_day (c, d1) != @ && customer_depart_day (c, d2) != @  &&
       reservation_ok (c, d1, d2) == @ && (∀n, room_availability (n, d1, d2) == "reserved")
         => reservation_ok (c, d1, d2) = FALSE
}
```

It should be noted that these definitions are personal and subject to revision during the enaction of EM.

Now, the modeller can create a computer model corresponding to the LSD account. A general rule is to replace derivates by definitions and protocols by actions. For example, the following Eden definitions are given:

```
all_rooms_availability is [roomAV_101, roomAV_102, roomAV_103, roomAV_104,
                           roomAV_201, roomAV_202, roomAV_203, roomAV_204,
                           roomAV_301, roomAV_302, roomAV_303, roomAV_304];
roomAV_101 is check_room_availability(1, d1, d2);
roomAV_102 is check_room_availability(2, d1, d2);
…
roomAV_304 is check_room_availability(3, d1, d2);
```

Also, visualisation using the observation tools is taken into account during the creation of the model. For example, a Scout screen corresponding to the reservation table is created as shown in Figure 2-3.

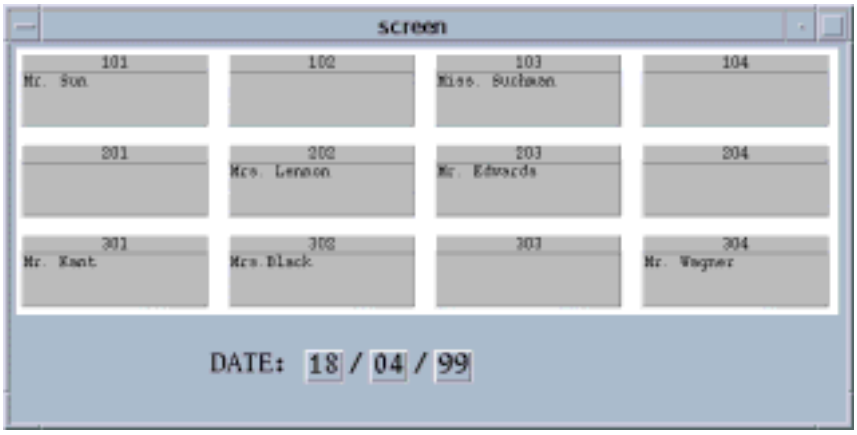

Figure 2-3. A snapshot of the computer model for the hotel booking system

To animate the process of making a reservation as described above, the modeller can select a date and a room slot and then enter the customer's name. After the interaction, it is found that data lists in the form of a reservation book used by the receptionist are needed in order to save the entered data. Hence, the following definitions are given[16]:

```
Year99 is [Jan99, Feb99, Mar99, Apr99, May99, Jun99, Jul99, Aug99, Oct99, Nov99, Dec99];
Jan99 is [day010199, day020199, …, day310199];
Feb99 is [day010299, day020299, …, day280299];
…
Dec99 is [day011299, day021299, …, day311299];
day010199 is array(12);
…
day311299 is array(12);
```

Thus, the modeller can create procedures to store the customer's name in the data lists above when the name is entered into a room slot for a specific date.

Moreover, it is found that identifying the reservation table as an agent is helpful in introducing automatic checks on data integrity. For example, the agent's protocol can be configured to prevent the receptionist from mistaking the availability of a room slot when

[16] The current EM tools do not support the functionality of a database, so data manipulation is difficult. This so far is a limitation of EM (see Section 3.3). For the sake of simplification, the details of data manipulation are omitted here and in most examples given in this thesis.

making a reservation. An appropriate LSD account for the reservation table agent is defined as follows:

```
agent reservation table (d1, d2) {
  state
    room_slot (n,d)  /* the availability of room n at date d */
    room_availability (n, d1, d2)  /* the availability of room n during the period of d1 and d2 */
 handle reservation_error  /* the same room slot is allocated to different customers */
 derivate
  room_availability (n, d1, d2)
            = (∃d, d1<=d<d2, room_slot (n, d ) ! = "" ) ? "reserved" : "available"
 protocol
  room_slot (n, d) != c && reservation_ok (c, d1, d2) != TRUE && input_roomslot (n, d, c) == TRUE
                          => reservation_error = TRUE;
}
```

At the same time, for the LSD account of the receptionist agent, the definition of the observable *room_availability(n, d1, d2)* is removed and the following protocol is given in response to the new added agent.

```
handle input_roomslot(n, d, c) /* the receptionist intends to input the cusomer name c into room slot n for day d */
protocol
  reservation_error == TRUE   =>  input_roomslot (n, d, c) = FALSE;
```

The decision made by the modeller to identify a reservation table agent indicates that the modeller views making a reservation and the rules of making a reservation as conceptually distinct.

The modeller can continually use the computer model to explore, expand and experience his/her understanding of the intended system through 'what if' experiments for diverse purposes. For example, the following experiments have been conducted:

1. What if each of the room slots is coloured to indicate its availability?
2. What if the customer changes his/her arrival date?
3. What if the receptionist needs to reallocate one or more reserved rooms to make optimal use of the hotel's accommodation?

4. What if the customer asks for information about a room, such as price, bedding style, windows' orientation, and so on?

5. What if a reserved room slot is re-reserved for another customer?

Each experiment could change both the LSD account and the computer model. In the case of experiment 1, the following is added into the LSD account of the reservation table agent:

```
state room_slot_colour (n, d1, d2)
  derivate
    room_slot_colour (n, d1, d2) = (room_availability(n, d1, d2) = = "available" )? "grey" : "yellow"
```

In addition, the computer model is modified for invoking experiments 1 and 4. Figure 2-4 shows a snapshot of the revised computer model.

In the same manner, more experiments can be invoked by means of definitive programming for improving the understanding of the intended system. This understanding helps to maintain the virtual correspondence between the computer-based model and the system used by the receptionist in the real world. In most cases, the exploratory and interactive process of enacting EM must be stopped at some moment in order to deliver the developed system to the user. However, the openness of the developed system can still be persistent, and this in turn allows the user as the modeller to continue the enaction of EM during the use of the system in the real world. (More details are provided in the next chapter).

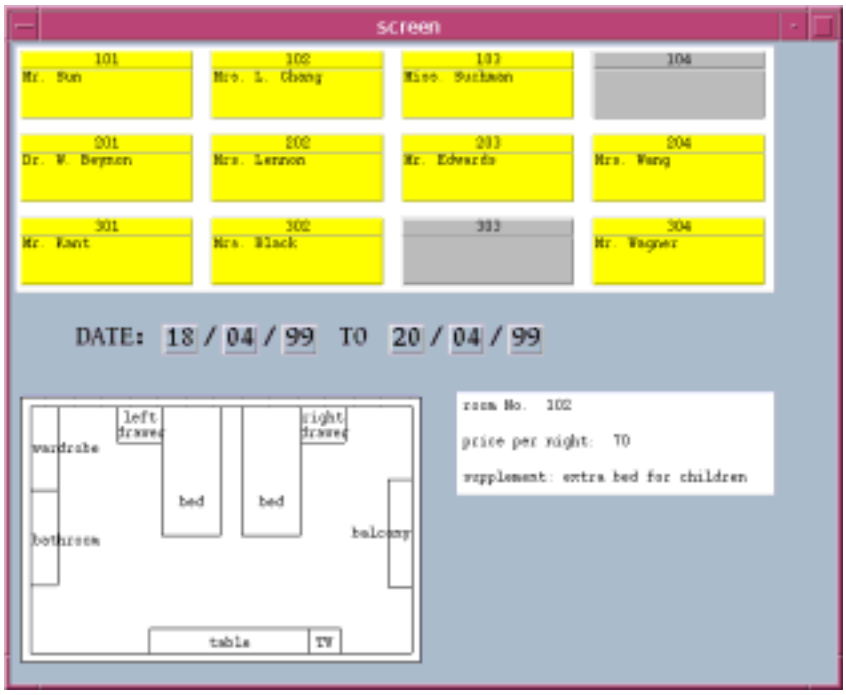

Figure 2-4. A snapshot of the computer model for a hotel booking system after further experiments