2 Empirical Modelling

In this chapter, we shall give the reader an overview of EM principles and tools. The contents of this chapter are organised in such a way that it gradually moves from abstract concepts to concrete implementations. Section 1 describes the underlying philosophy of EM. Section 2 introduces the modelling framework that is based on the underlying philosophy. Section 3 explains how the framework is realised in practice.

This chapter should not be regarded solely as a literature review. Since the concepts of EM itself have been evolving over the past 20 years or so, this chapter can be regarded as a consolidation of concepts introduced in the EM project. In particular, we gather the EM concepts into a unified and coherent framework (the Definitive Modelling Framework) that has characteristic concepts for agent representation (section 2.2.1) that support modelling activities (section 2.2.2) based on a distinctive interactive style of model building (section 2.2.3).

2.1 A commonsense understanding of phenomena

The philosophical foundation of EM is based on the way we understand and interact with the world. In this section, we shall discuss the main concepts associated with it. These concepts are *not* new in the sense of having been newly invented. On the contrary, they originate from a commonsense way of understanding phenomena in the world that we experience in everyday life. EM privileges these commonplace concepts and provides a framework to embody them into computer-based artefacts – to help the cognitive process of understanding phenomena. The discussion in this section is based on Beynon's exposition of concepts of EM in [Bey02a, Bey02b].

2.1.1 Concurrency as experienced

Central to EM concepts associated with understanding phenomena is the notion of *concurrency*. In classical computer science, the notion of concurrency usually involves several processes executing in a common environment. Each process is a list of sequential steps of computation that specify operations and interactions associated with predefined states and behaviour. The focus of study is on designing specification notations for constructing abstract mathematical models to represent the behaviour of concurrent systems (e.g. CSP [Hao85] and CCS [Mil89]).

In EM, our concern in thinking about concurrency is much broader than that of traditional computer science. We are concerned with a notion of concurrency *as experienced* in everyday life through the interaction of human agents, natural forces and artefacts. As pointed out by Beynon [Bey02b], concurrency as experienced is reflected in the dictionary definitions:

> *Concurrent – running, coming, acting, or existing together*
> *Concurrence – joint action, coincidence, assent*

All the definitions here are meaningful relative to an observer with reference to time. Beynon describes three significant features of this observation:

- Subjectivity with respect to what exists together – what exists together is purely an interpretation of phenomena in the mind of an external observer. The mental conception of what exists together is unique and personal to different observers even in the same context.

- Discrimination in identifying observables and agents – in the process of construing what exists together, an observer presumes an identification of separate entities and their coincidence. After considering the existence of dependencies among observables, certain state-changes to observables are attributed to groups of coexistent observables that can be construed as agents.

- Discretion over viewpoint and mode of observation – an observer has some degree of control over what he or she wants to observe and ignores others.

In this context, the classical notion of concurrency is relevant only when issues associated with subjectivity, discrimination of essential entities and discretion over viewpoint can be resolved in an unambiguous way. This is the time when the observer has a full and precise understanding of the phenomena in question. Therefore, classical formal specification notations are suitable for representing the end-result of the understanding. In contrast, EM privileges powerful concepts (namely *observation*, *agency* and *dependency*) derived from the analysis of commonsense concurrency outlined above, and provides tools to assist the process of understanding before a possible end-result may eventually emerge.

We shall discuss the concepts of observation, agency and dependency respectively in the next three subsections.

2.1.2 Observation

Observation plays a key role in understanding phenomena. Observation is typically associated with our sensory experience of the environment through vision, hearing, touch, taste and smell. In EM, the concept of observation is extended to embrace determinants of state that can be directly apprehended by the observer whether or not they are sensory in nature. On this basis, observation can be purely abstract in that it involves only exploration of our own thoughts. In fact, observables are anything that can be given an *identity*. The identification of observables is a matter of the observer's interests. Therefore, *existence* of observables is subjective in nature.

From this subjective perspective, the observables that are present can be regarded as defining the observer's 'current state of mind'. In explaining this, Beynon [Bey02b] likens a state of mind to a physical location in the real-world, in which the observer can dwell, from which he can leave, and to which he can return. The current state of mind is associated with the observer's current focus of interest. A change in the current state of mind may be linked to a change in the observer's current focus of interest. A *state-change* can be the result either of an action on the part of the observer (either voluntary or involuntary and at the conscious or the subconscious level) or an action on the part of other agents in the situation.

When it comes to understanding complex phenomena with concurrency (e.g. understanding the complex behaviour of a concurrent system), the observer's state of mind can be overloaded in the sense that there are too many observables at different levels of abstraction that have to be considered at the same time. In this case, the

conceptual integrity of the complex phenomena is difficult to maintain – so sense making becomes difficult. EM addresses this issue by seeking to embody and manage the state of mind of the observer using a special-purpose computer-based artefact.

2.1.3 Agency

Agency is a commonplace concept in our informal understanding of phenomena and interactions with the world. It is associated with *attributing state-changes* to what is understood to be their primary observed source. The informal nature of agency as perceived presents a challenge for conventional approaches that try to formalise agency or even factor out agency (as in a formal notation such as Concurrent Sequential Processes (CSP) that focuses on processes rather than agency as its primary abstraction in studying concurrency).

A natural question to ask at this point is: what entities in the world can be viewed as agents? This question cannot be answered without making reference to a particular context, and more importantly it can be answered only with reference to the state of mind of some observer. Agency is shaped by the subjective requirements, private interests and previous experience of the external observer. A change in the value of an observable can arise in a wide variety of ways.

In our informal interaction with the world, there is *no* single characterisation of the agency concept. In this respect, *commonsense agency* differs in nature from the two notions of agency widely accepted in the agent-oriented research community [Woo95]. In the *strong* notion of agency, an agent should be explicitly defined in terms of cognitive concepts such as beliefs, desires and intentions. In the *weak* notion of agency, an agent should be defined in terms of the observable properties that it exhibits, such as autonomy, reactivity, pro-activity and social ability. Both notions of agency try to formalise the agency concept and reduce it to a set of universal definable properties. They are arguably both inappropriate for the construal of agency in the context of commonsense concurrency.

It is difficult to discriminate between one entity as an agent and another. In fact, even a primitive observable can be an agent in a broad sense (e.g. “Ouch! This wooden spike is hurting me!”). Commonsense agency has more in common with what Lind calls the *very weak* notion of agency where indeed *anything* can be an agent [Lin00]. In the context of system design, Lind explains that:

> *"…the conceptual integrity that is achieved by viewing every intentional entity – be it as simple as it may – in the system as an agent leads to a much clearer system design and it circumvents the problem to decide whether a particular entity is an agent or not." [Lin00]*

This quotation highlights the pragmatic advantage of adopting the very weak notion of agency. But the conceptual advantage of avoiding a prescriptive notion of agency is even more important – if we make a conscious effort to identify 'sources of state change' as we experience them in our everyday life, we find that they suggest three quite different views of agency:

- Primitive agency: agents as associations of observables. When considering the sources of state change, we commonly identify particular groups of observables as constituting a single agent. One significant characteristic that guides such grouping of observables is 'existence dependency' – sets of observables are observed to be present or absent at the same times, and so 'act as one'. Even if we have no direct experience of how such association of observables can effect state change, their presence or absence alone typically influences the perceptions of the observer. This leads us to regard such associations as agents in so far as they can potentially be the cause, cue or trigger for some action on the part of another agent. By this criterion, every observable is an agent and so is the external observer.

- Explicit agency: agents as the perceived sources of open-ended state change. A primitive agent can be viewed as an explicit agent when empirical evidence leads us to attribute state changes to it. Identifying explicit agency is typically associated with growing familiarity with a particular context for observation on the part of the observer, and with a degree of specialisation of observation towards a specific focus of interest or goal. The attribution of state change does not normally entail assumptions concerning the circumstances in which it occurs or what stimulus might be responsible. Such issues will typically be the subject of ongoing experiment.

- Circumscribed agency: agents as actors with circumscribed behaviour. In some circumstances, the attribution of agency and the context for interaction can be so precisely identified or prescribed by the observer that the agent can be interpreted as acting like the actors in a play or the components of a system. Identifying circumscribed agency is generally

> associated with a very high degree of specialisation of observational context that involves strong presumptions and discretionary constraints concerning acceptable interaction on the part of the observer. Like real world actors in a play, circumscribed agents are in significant respects no longer autonomous in the way that explicit agents can be. They are 'virtual' agents in the closed-world, where the context of observation is so circumscribed that (as far as the observer's specific viewpoint is concerned) nothing new can be learnt from further study of the phenomenon that is the subject of interest.

As we understand phenomena in the world, our view of agency has a tendency to progress from the primitive through the explicit to the circumscribed. EM provides a framework to accommodate all three views and is particularly concerned with explicit agency, which is somewhere between the primitive view, where the agency is almost vacuous, and the circumscriptive view where it is very tightly constrained.

2.1.4 Dependency

Dependencies are patterns of stimulus-response between observables with some degree of persistency. Identifying dependencies among observables also plays a key role in understanding phenomena. To identify dependencies, the observer needs to have a perception of state-change and identity (e.g. to see what moves), combined with the ability to remember and so to have an *expectation* (e.g. to predict what will move), and to correlate the action with the state-change (e.g. to know what causes the move). Typically, such identification is associated with a prolonged period of observation and/or interaction with the phenomena that may be either deliberate or accidental. Activities that lead to the identification of dependencies by the observer of a phenomenon can be categorised as follows:

- Pure observation – The observer either has no way to, or intentionally does not, influence the phenomena. For example, cosmology is entirely founded on pure observation of a system over which we apparently can exercise no direct influence.

- Direct interaction – In some cases, direct and directed intervention is possible. The observer can interact with the phenomena in the role of experimenter.

- Off-line experimentation – The observer may be able to carry out 'off-line' experiments to test particular hypotheses about stimulus-response patterns

without directly interacting with the phenomena.

In all cases, dependencies originate from past experience and are confirmed by the present observation and/or interaction. The purpose of identifying dependencies is to inform future expectations of events associated with the phenomena. In all three activities, the capacity of the observer to be surprised by responses is particularly important. Surprises typically reveal subtleties of phenomena that require further investigation and may eventually lead to the discovery of new observables, agents and dependencies. In this respect, being able to recognise explicit agency is significant.

2.2 The Definitive Modelling Framework (DMF)

In this section, we introduce the Definitive Modelling Framework (DMF). The DMF is a framework for building artefacts that embodies understanding of phenomena in terms of commonsense notions of concurrency, observation, agency and dependency. It is a 'modelling' framework because it is primarily associated with building artefacts; it is 'definitive' because the modelling process involves exploring *definitions* that form an important part of an artefact. The term 'Definitive Modelling Framework' rather than 'Empirical Modelling Framework' is adopted on the basis that there are arguably modelling activities that involve embodying observation, agency and dependency in artefacts that do not explicitly use the computer-based representations to be described below. The main aim of the DMF is to assist the cognitive process of understanding phenomena. The introduction of the DMF serves to consolidate EM concepts developed in the past 20 years or so that include the Abstract Definitive Machine (ADM) [Sla90], an EM environment for concurrent engineering [Adz94a] and Interactive Situation Models (ISMs) [Sun99a].

2.2.1 The representation of observables, dependencies and agents

In the DMF, observables and dependencies can be represented as *definitions*. A typical definition takes the form:

$$y \text{ is } f(x_1,x_2, \ldots , x_n)$$

where $y, x_1, x_2, \ldots , x_n$ are *definitive variables* (or *variables* for short), *is* is a separator and *f* is a *formula*. A definition is divided into two parts separated by the *is* separator,

viz. its left-hand-side (LHS) and its right-hand-side (RHS). The LHS of a definition typically contains only one variable (*y*) whose value is determined by the formula on the RHS. The formula contains algebraic operators and functions that reference a set of variables (and possibly some constants) as operands and is used to calculate the current value of the LHS variable (*y*). The purpose of the formula is similar to that of the formulae in the cells of a spreadsheet.

A definition is a representation of an *indivisible stimulus-response pattern* (or a dependency) between observables. Observables are represented by variables. The current status of observables is represented by the current values of variables. The maintenance of the stimulus-response pattern is specified by the formula – 'stimulus' is associated with any change of value in the variables referred at the RHS; 'response' is associated with an update to the value of the variable at the LHS. The term 'indivisible' is used to convey the idea that the maintenance of the relationship established by the stimulus-response pattern permits no external interruption.

In the DMF, a set of definitions provides the fundamental representation of state. The primary role for such a set of definitions is similar to that of the set of definitions underlying a spreadsheet: it supplies values and dependencies that correspond closely to the observables and dependencies in the situation it represents. A set of definitions can also be used to model behaviours. For this purpose, subsets of definitions are grouped into *entities* that correspond to primitive agents in the environment. The current state of the environment is determined by the primitive agents that are present typically together with generic persistent observables such as time and gravity; in the DMF, a set of definitions that represents state accordingly comprises groups of definitions within an entity that come and go together (in view of their existence dependency), together with isolated definitions that are typically persistent.

With this model of state, the effect of all kinds of agent actions can be expressed in the DMF. A typical action takes the form:

$$g(x_1, x_2, \ldots, x_n) \rightarrow (\ \textit{variable redefinition} \mid \textit{entity creation} \mid \textit{entity deletion}\ )^*$$

where $x_1, x_2, \ldots, x_n$ are variables, *g* is a *guard*, and the action consists of a sequence of operations to *redefine variables*, and *create* or *delete* of *entities*. The guard is a logical expression that references a set of variables (and possibly some constants) as operands. An action is a representation of a latent interaction or experimental action associated with some agent. When the guard evaluates to true, the agent who owns

this action has the privilege to perform the sequence of redefinitions, creations and deletions specified at the right-hand-side of the action. Note that the fact that the agent has the privilege to act does not necessarily mean that the agent has to act.

Definitions and actions equip the DMF with the mechanisms needed to support our commonsense construal of phenomena. Their expressive power stems from the intelligent engagement of the human observer in their interpretation and execution. The appropriate way to represent an explicit agent in the DMF is by an entity that comprises a group of definitions together with an associated group of latent actions. By default, all action is conceived as performed by the human observer. Only the human observer can make due allowance for provisional and imperfect knowledge of the conditions governing action. Only the human observer can play the part of a super-agent, contributing actions that are not preconceived such as representing experimental interactions and random events.

When using the DMF for modelling and simulation, it is not practical or appropriate for the observer to be responsible for all actions. Some actions are typically executed automatically as in a machine. The extent to which automatic execution is meaningful depends upon the extent to which the activities of explicit agents can be circumscribed. In general construal and simulation of concurrent behaviour within the DMF, both human and machine execution are represented. What is more, whether such behaviour is to be understood as design, use or testing activity is a matter of interpretation.

Figure 2.1 depicts the *dual perspective on agency* in the DMF. From the *human perspective*, an agent can be viewed in terms of the observables it owns, identified dependencies among the observables (i.e. indivisible stimulus-response patterns), and latent interactions and experiments. From the *machine perspective*, an agent is represented by an entity that consists of a group of definitions, possibly together with a group of guarded actions.

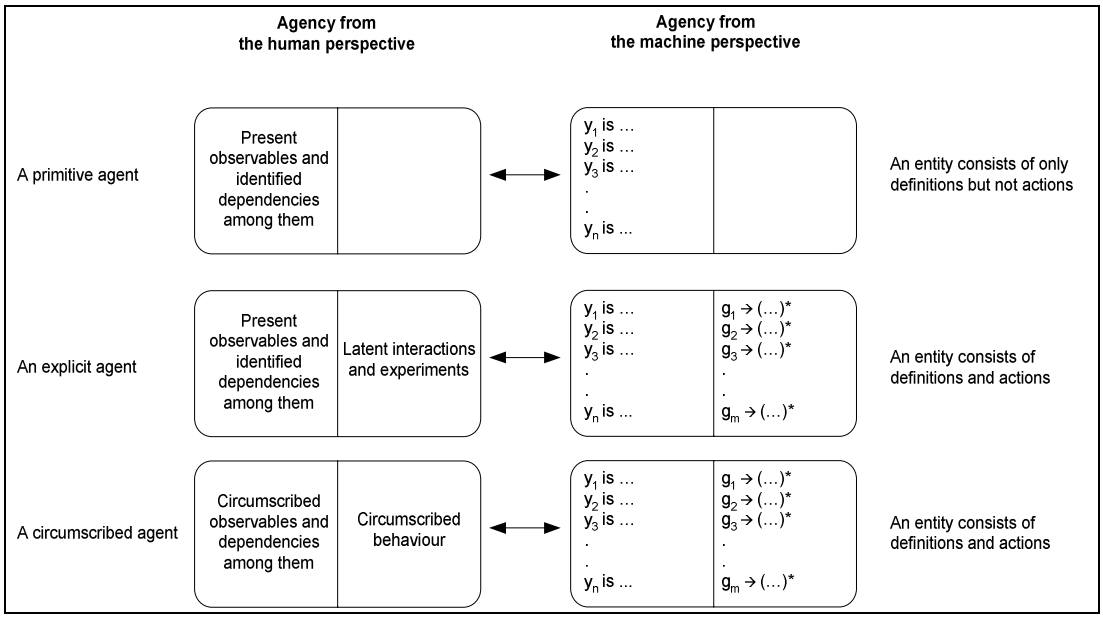

Figure 2.1: The dual perspective on agency

In the DMF, any agent can be described from *both* the human and the machine perspective whenever they are appropriate. It is important for the external observer to keep both perspectives in mind when analysing agencies. When we consider the absolute discretion that is involved in human execution of actions, the significance of the definitions and actions attached to an agent in the machine perspective is called into question. Human execution can discount these definitions and actions entirely. Viewed in this light, the purpose of representing agency from the machine perspective is to provide a representation of what is already known but provisional; in contrast, the human perspective encourages exploration of what is still unknown. On this basis, the duality facilitates the representation of provisional knowledge about the context in which agent actions are performed and the acquisition of knowledge of a similar nature that is still unknown.

The motivation for the dual perspective is similar to that explained by McCarthy in his discussion of "ascribing mental qualities to machines" [Mcc79]. McCarthy points out that it is sometimes useful to ascribe certain mental qualities like beliefs, intentions and wants to a machine. In the context of understanding the behaviour of a program, he describes the reason for ascribing mental qualities as epistemological:

> *"… ascribing beliefs is needed to adapt to limitations on our ability to acquire knowledge, use it for prediction, and establish generalizations in terms of the elementary structure of the program."*

[Mcc79]

Our concern is to understand phenomena in general rather than to understand the behaviour of specific programs. However, in the DMF, the rationale for being able to treat any agent as human-like is similar to McCarthy's reason for ascribing mental qualities to machines.

Figure 2.2 shows a representation of multiple agents identified in the process of understanding a phenomenon. We can see that there is a circumscribed agent whose behaviour has been well understood; there is an explicit agent that has yet to be – and may never be – circumscribed; there are primitive agents associated with sets of variables with integrity, and with definitions outside the boundaries of all agents. In the picture, a solid outline around an agent means that the boundary of the agent is fixed; a dotted outline means that the boundary of the agent is subject to further revision. This convention will be used to depict the various kinds of agency throughout the discussion of the DMF in this chapter.

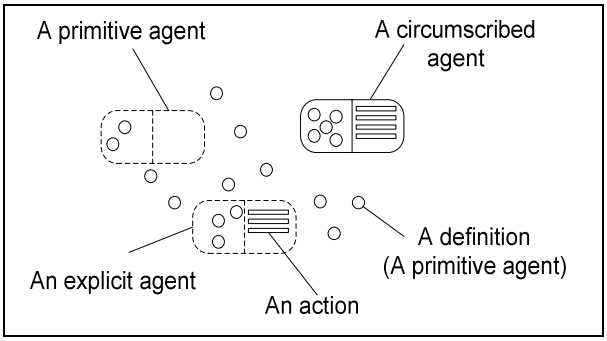

Figure 2.2: A representation of multiple agents

The significant features of the representation of multiple agents within the DMF can be summarised as follows:

- Concurrency – There are two main types of concurrency. Firstly, different agents can perform actions concurrently. Secondly, the dependencies associated with definitions are maintained concurrently.

- Indivisible stimulus-response patterns – Definitions represent dependencies among observables whose evaluations permit no external interruption.

- Dual perspective on agency – Any agent can be viewed as a human-like agent.

- Openness of privileges – The representation imposes *no* constraint on how the observables/variables owned by an agent are referenced by others.

In the next subsection, we shall discuss how modelling activities of EM can be performed using the DMF.

2.2.2 The modelling activities

The representation of multiple agents described above serves as an artefact that embodies our understanding of the phenomenon. To understand the modelling activities under the DMF, we need to conceptually distinguish between four entities: the *modeller*, the *phenomenon*, the *referent* and the *artefact*. Figure 2.3 depicts the relationships between them. The artefact embodies general experience of the phenomenon. The referent is the aspect of the phenomenon that is of particular interest to the modeller, and in general evolves to reflect the specialised ways of interacting with and interpreting interaction with the artefact that develop in the modelling process. The modeller as the external observer is the archetypal agent who makes the fundamental and essential semantic link between the referent and the understanding of it (as embodied in the artefact he builds). Only the external observer has the capacity to be surprised by responses from the referent or the artefact. Surprise can lead to a shift in perspective that prompts a point of departure from what has been previously explored, and leads to a deeper understanding of the referent and an enrichment of the artefact. We shall discuss the importance of using an artefact, typically computer-based, to represent our understanding in more detail in the next subsection.

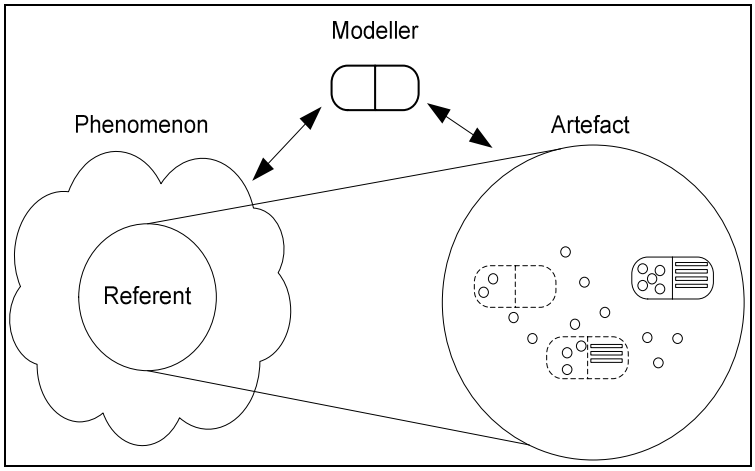

Figure 2.3: Relationships between the modeller, the phenomenon, the referent and the artefact

There are two important and interrelated modelling activities namely *agentification* and *role-playing*. Agentification involves identifying agents in the referent and analysing the observables to which they may be construed to respond. Figure 2.4 depicts the typical pattern of progress in the agentification activity over time. Initially, only a few observables and dependencies have been identified by the modeller. The view of agents is at its most primitive – all observables can be interpreted as primitive agents. As the modeller obtains more understanding about the referent, some explicit agents are identified as being responsible for state-changes to some observables. An element of shift in observational perspective on the part of the modeller is also involved here. Gradually, deeper understanding shapes some circumscribed agents whose behaviour is so predictable that they cannot surprise the modeller at all. This is a creative and experimental process. There is *no* universal generic procedure to follow through which successful representation or understanding is guaranteed. Success depends upon being able to correlate state-based observations in order to identify stimulus-response patterns by interacting and experimenting with the artefact as well as the referent.

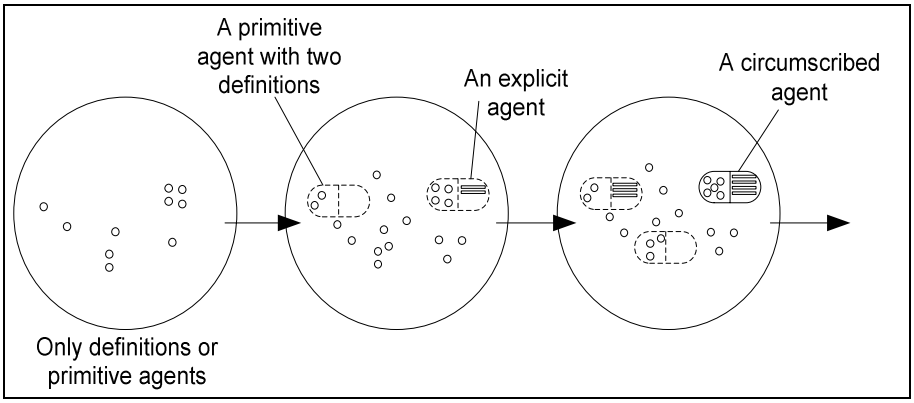

Figure 2.4: Agentification as progression from primitive to explicit and circumscribed views of agency

The concept of role-playing activity is a feature of EM that has been discussed by several previous researchers (cf. [Sun99a, Run02]). This involves a conceptual distinction between different roles that the modeller can play. The modeller can be considered as a *super-agent* who can play the roles of the *external observer*, *acto*r and *director* described as follows:

- The modeller as the external observer – The modeller interprets what is going on in the artefact with reference to his observation and perception about the

phenomenon. There are many factors that can affect the modeller's interpretation which include his past experience, future expectation, knowledge of current interactions, subjective judgement, and even his physical ability. Observations and their interpretation contribute to a deeper understanding of behaviour of both the artefact and the referent.

- The modeller as an actor – The modeller construes a situation from the perspective of an internal agent who is directly interacting with other agents. When the modeller acts in the role of an actor, she is not only able to see things from the perspective of the actor, but is also in a position to reflect upon the actor's role in its overall context.

- The modeller as a director – As a director, the modeller directs the internal agents in the execution of their actions. This activity is similar to directing a play or simulating the execution of a concurrent program. In this case, the modeller can experiment by executing, modifying and interrupting existing actions, or by introducing new actions.

Figure 2.3 depicts the relationships between the modeller, the phenomenon, the referent and the artefact in an abstract and simplistic way. The figure is helpful to the reader in understanding their significance in the modelling activities, but does not always convey the full nature of the phenomena and the referents that arise in practice. When we consider the modelling context for system development in more detail, we can distinguish three scenarios (see Figure 2.5):

- Scenario 1: the artefact represents a fixed referent – In this scenario, the modeller constructs the artefact by observing a specific aspect of the phenomenon as its given fixed referent. Interactions with the phenomenon are primarily aimed at obtaining more understanding of the referent rather than changing it. The key activities involved in this scenario are *analysis* and *simulation* of an existing phenomenon that is at some level well-understood, as for example, in creating a football game simulation [Tur00]).

- Scenario 2: the artefact and its referent co-evolve – The aspects of the phenomenon that are of interest change as the artefact is developed. Thus, changing the artefact affects the referent. The modeller is not only observing the phenomenon but also shaping the referent. The key activity involved is the *design* of the artefact that contributes to the discovery of new aspects of the

phenomenon (cf. [Loo98]), as for example, in the design of a digital watch [Roe01].

- Scenario 3: the phenomenon comprises model-building activities – The modelling involves treating modelling activities performed by some other modellers from a higher viewpoint. The meta-modeller can be thought of as a coordinator whose purpose is to *negotiate meanings* with other modellers within the phenomenon. The referent (not depicted in Figure 2.5) is associated with those aspects of the interaction between other modellers that concern her in the role of coordinator. These might include issues such as requirements and deadlines imposed upon modellers, work patterns for their collaboration and key characteristics of their artefacts.

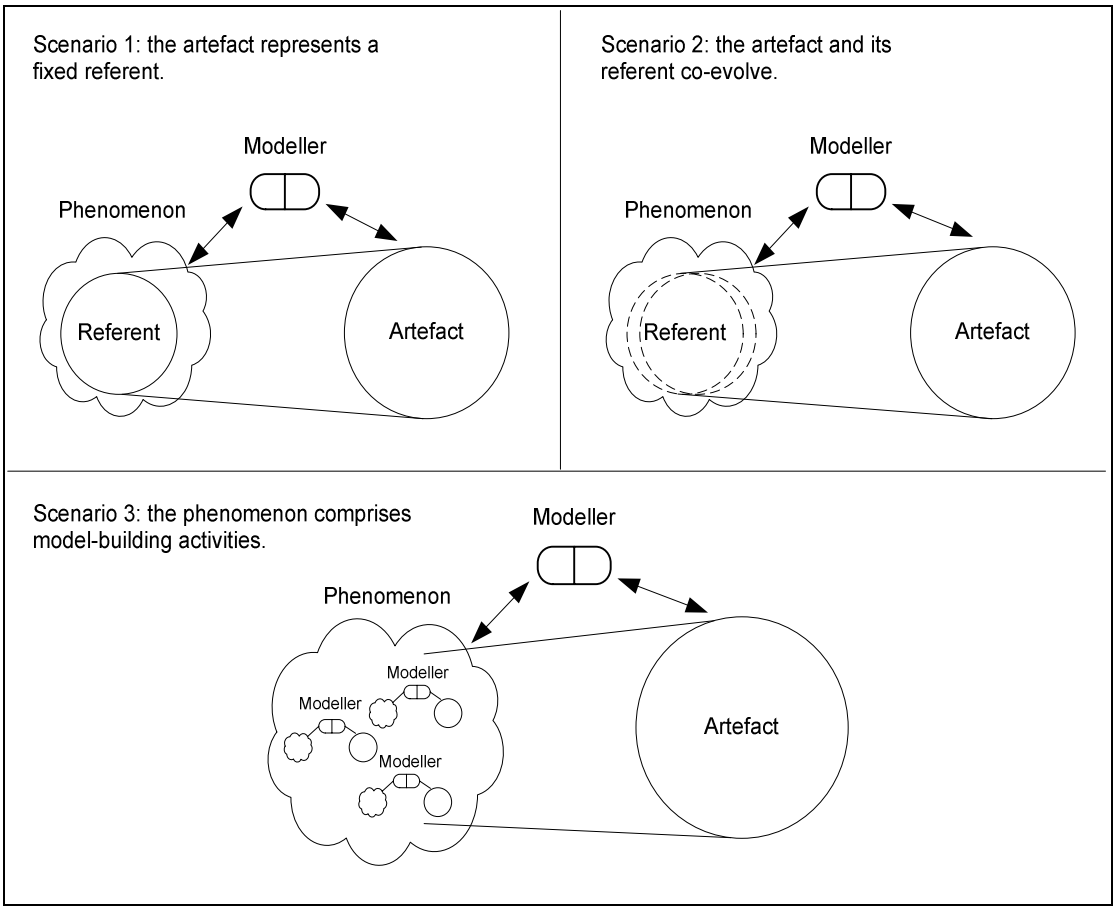

Figure 2.5: Scenarios of relationships between the phenomenon, modeller and artefact

Scenario 3 conveys the idea of *collaborative definitive modelling* in the DMF that requires some further elaboration. In the context of applying EM to concurrent engineering, Adzhiev et al. [Adz94a] introduce the concept of a hierarchy of designer agents who collaborate with each other in designing engineering products. This

concept can be generalised or reinterpreted to apply to all kinds of definitive modelling that involve a team of human-like agents, i.e. a team of modellers. Collaborative definitive modelling involves developing interpretations and negotiating and sharing understanding of a referent among a team of modellers. The current status of the *consensus* can be understood with reference to a hierarchy of modellers, as shown in Figure 2.6 below.

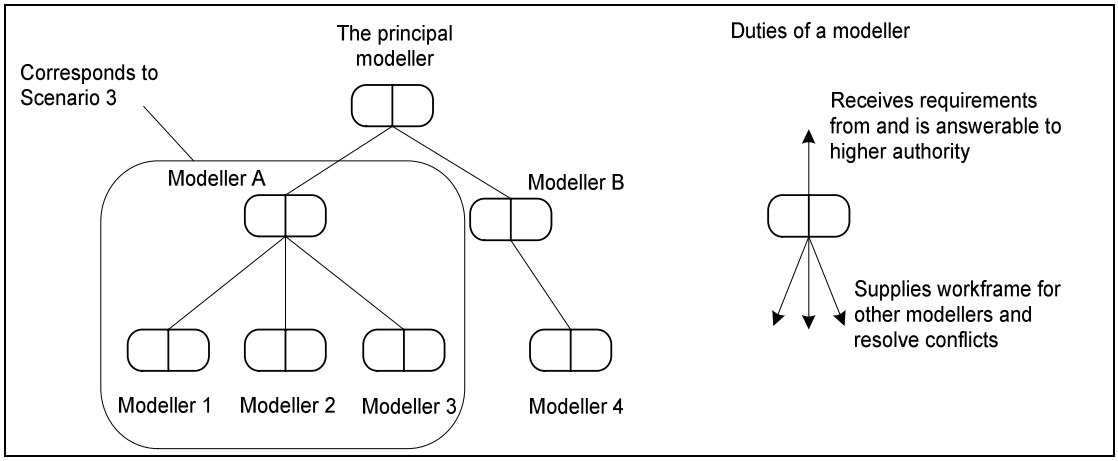

Figure 2.6: A hierarchy of modellers in collaborative definitive modelling

The hierarchy of modellers is actually a generalisation of Scenario 3 in Figure 2.5. Each modeller internal to the hierarchy has two roles. As a coordinator, a modeller supplies a frame for the work of other modellers and resolves conflicts. As a worker, a modeller receives requirements from and is answerable to a higher authority, that is, act as a modeller at a higher level of the hierarchy. At the upper-levels of the hierarchy, the modelling activity has a managerial aspect that involves framing the design objectives and patterns of interaction between the modellers at the next lower levels. At the lowest level of the hierarchy, the modelling activity focuses more on working on specialised experiment and understanding that impacts on specific aspects of the whole consensus.

The consensus in collaborative definitive modelling is evolving all the time according to emerging understanding of the referent. Each modeller builds an artefact to help his understanding of certain aspects of the phenomenon that contributes to the whole consensus. This interaction of minds is in the same spirit as Minsky's agent hierarchy in his Society of Mind [Min88]. The artefact also serves as a medium for sharing knowledge. This hierarchical framework for collaborative definitive modelling expresses the potential conceptual integrity of the intermediate understanding of the referent, which has a potential for consensus that encompasses

every modeller in the team. This issue will be further discussed in chapter 5.

We summarise the significant features of the modelling activities in the DMF as follows:

- State-based observation – It is important for the modeller to be able to directly observe and experience states of the referent and the artefact so that he can correlate them and identify patterns among them.

- Evolutionary construal – Agentification is an evolutionary learning process typically starting from identifying primitive agency to shaping explicit agency and eventually but not necessarily proceeding to specifying circumscribed agency.

- Role-playing – The modeller can play several different roles (external observer, actor, director and coordinator) to effect the shifts of perspective necessary to acquire deeper understanding.

- Subjectivity – Modelling as a personal affair is represented in the DMF. Subjective interpretations and interactions are central to the concerns of understanding phenomena.

- Collaboration – Modelling as a collaborative affair is represented in the DMF. The DMF has a potential to facilitate the collaborative creation and understanding of complex phenomena that involves a team of modellers (see chapter 3 and 5 for examples).

2.2.3 Interactive Situation Models (ISMs)

This subsection discusses the importance of the *interactive* nature of the artefact in the DMF. From the discussion in the previous subsection, we know that the artefact in the DMF plays several important roles:

- in representing the modeller's personal understanding (in terms of definitions, actions and agents);
- in the activities involved in acquiring new understanding (agentification and role-playing);
- in the animation of behaviour (through potentially automatable dependency

maintenance and action triggering);

- in communication between modellers (collaborative definitive modelling).

These roles indicate that the artefact in the DMF is more than an abstract representation such as a formal specification might provide. The artefact actually facilitates the whole cognitive process of understanding and creating phenomena. To construct an artefact that can meet the demands of all the above roles, we need to use the computer. Indeed, EM is particularly concerned with *using the computer as an instrument* [Bey01a] that mediates between a phenomenon as conceived by the modeller and the phenomenon as perceived.

To emphasise that we are in fact concerned with a very special kind of artefact, we refer to the artefact built by using the DMF as an *Interactive Situation Model* (ISM), a term introduced by Sun [Sun99a]. An ISM is a computer-based artefact constructed through *situated* modelling activities. The use of term 'situated' reflects the significance of both the social and cultural context (cf. [Suc87]), and the specific physical environment in which the modelling is conceived [Bey98]. Unlike most computer models, which have a fixed interface and preconceived use, an ISM is 'interactive' in the sense that it is open to elaboration and unconstrained exploratory interaction. In something like the way that clay is suitable for modelling physical shapes, an ISM is suitable for modelling our conceptual understandings. By embodying patterns that reflect commonsense concepts of concurrency, observation, agency and dependency in an ISM, the modeller aims to establish an intimate relationship between the ISM and his or her evolving knowledge about the referent.

Building ISMs is closely related to the way in which experimental scientists use artefacts as a means of devising *construals* through observation and experiment. The term 'construal' was used by Gooding to refer to the concrete artefacts that embody insight into experimental interactions such as are described in [Goo90]. In particular, Gooding describes how Faraday, in developing his understanding of electromagnetic phenomena, constructed artefacts to represent observables such as electrical currents and magnetic fields, and dependencies such as the relationships between the polarity of a magnetic field and the direction of current. The importance of physical artefacts in supporting understanding is also endorsed by Feynman's view of what it means to be a physicist:

> *"a physical understanding is a completely unmathematical, imprecise, and inexact thing, but absolutely necessary for a physicist" [Fey64].*

In its essence, an ISM resembles what Gooding characterises as a construal. There is ambiguity concerning the extent to which a construal should be identified with a physical object or with a mental model. As Gooding observes in [Goo01] this ambiguity presumes a philosophical stance that is at odds with the traditional dualist separation between the mind and the physical world.

Throughout this thesis, we shall make no distinction between the terms 'ISM', 'EM model' and 'artefact'. They are synonyms that describe a model built by using the DMF but emphasise different aspects of the nature of the model.

2.3 EM in practice

This section describes two important techniques developed to support EM in practice. Firstly, we shall introduce an implementation of the DMF that is commonly used in EM, namely 'modelling with definitive scripts'. Secondly, we shall introduce a technique that has the potential to document the understanding gained from modelling activities, namely the LSD notation.

2.3.1 Definitive scripts

The main EM tool currently used for supporting modelling activities under the DMF is *TkEden*. As explained in more detail later in this subsection, to construct an ISM, the modeller can specify definitions by using some *definitive notations* to form a script of definitions, or *definitive script*. TkEden interprets the definitive script that is input by the modeller and introduces the definitions into the ISM. TkEden also *automatically* maintains the dependencies among definitions by keeping the value of every definitive variable up-to-date. This subsection only gives the brief explanation of constructing an ISM using definitive scripts that is needed for the reader to understand the illustrative models introduced throughout this thesis. For an extended treatise on modelling with definitive scripts, the reader can refer to [Run02].

The core of the TkEden interpreter is a simple interpreter, called TtyEden, that was originally developed by Y. W. Yung in his final year undergraduate project in 1987 [Yun90]. TtyEden has a text-based command line interface that supports interactive model construction in a single definitive notation called Eden (a general purpose language for definitive modelling). The tool was later extended by Y. P. Yung

in his study of the 'definitive programming paradigm' [Yun93]. In addition to Eden, Y. P. Yung integrated two other definitive notations into the tool: Donald (for 2D line-drawing) and Scout (for screen display). Since the tool makes use of the Tcl/Tk library for implementing windowing and graphical drawing, it is called TkEden. Subsequently, TkEden has been maintained and enhanced by several people in the EM research group, most recently and extensively by Ashley Ward who has developed and maintained an open-source distribution [Ope02]. Enhancements include implementing a prototype distributed version of TkEden (namely DtkEden developed by Sun [Sun99a]) and introducing other definitive notations (e.g. Sasami for 3D graphics and Eddi for database modelling [EMWeb]).

Figure 2.7 shows a screenshot of the TkEden interface. It displays four windows, namely the 'input window', the 'output window', the 'history window' and a 'definition window'. The 'input window' is where the modeller can redefine, delete and create definitions by writing definitive scripts. The 'history window' records the sequence of commands that have been input by the modeller. The 'definition window' shows some of the definitions that reside in the ISM: other windows to display complementary definitions can be accessed via the View option in the 'input window'.

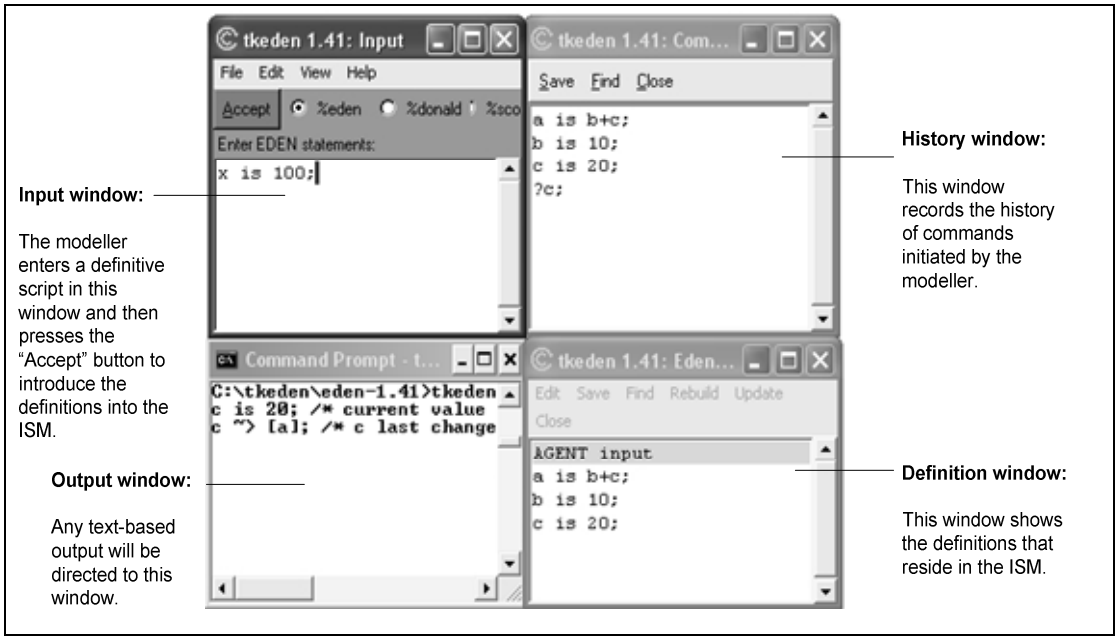

Figure 2.7: A screenshot of the TkEden interface

A definitive script may contain definitions in several different definitive notations. The term 'definitive notation' was first introduced by Beynon [Bey85] to

refer to a simple language that can be used to specify definitions in formulating a definitive script. The basic semantics of a definitive notation is determined by an underlying algebra of data types and operators but the syntax of definitions can take a variety of forms. For example, a definition to establish the following dependency:

"the value of the variable A is the sum of the values of the variables B and C"

can be formulated in TkEden in three different ways:

- `A is B + C;`
- `A = B + C`
- `A = B + C;`

A definitive notation can be specifically designed for a particular application domain to facilitate ease of use by using domain-specific terms and syntactic constructs. For the most part, the models illustrated in this thesis are constructed by using Eden, Donald and Scout in TkEden. The three illustrative definitions above are in Eden, Donald and Scout respectively. Some similarity with these notations is helpful in understanding the models introduced later in this thesis.

Eden

Eden (the Evaluator for DEfinitive Notations) was first designed and implemented by Y. W. Yung [Yun88]. The Eden notation is a general purpose language that implements the DMF concepts of definitions and actions. The Eden syntax and data types are similar to those in the C language. The basic programming constructs include `for`, `while` and `if`. The basic data types include `integer`, `float`, `string` and `list`. In fact, Eden is a 'hybrid' programming language that allows both definitive modelling and procedural programming to be performed in the same environment. The modeller can even specify their own *functions* that can be used in specifying definitions. This makes Eden extremely expressive but, to make most effective use of the tool, the modeller has to be conscious of the need to be as faithful as possible to the principles of EM in the DMF.

Significant features of Eden are described with illustrative examples as follows:

- syntax for definitions – the sample definitions shown below specify the dependencies between three variables. Note that each definition has its LHS and RHS separated by the `is` separator, and each definition is terminated by a

semi-colon '`;`'. There is no need to declare a variable before its use. The type of a variable is determined by the interpreter automatically. The dependencies between variables specified by the definitions are also maintained automatically. In this case, if the value of `b` or `c` has changed, the value of `a` will be updated according to its formula.

```
a is b+c;
b is 10;
c is 30;
```

- difference between a variable assignment and a definition – the two statements below are semantically different. The first statement specifies a definition but the second statement specifies an assignment. The first statement specifies that the value of `a` is *always* dependent on the sum of values of `b` and `c` so that `a` gets updated every time the value of `b` or `c` has changed). The second statement specifies that, unless and until it is reassigned, the value of `a` is equal to the current value of the sum of `b` and `c` at the point of assignment. In this case, the calculation is one-off, so that subsequent changes of `b` and `c` do not have any effect on the value of `a`. We refer to `a` as a definitive variable in the first statement and a procedural variable in the second.

```
a is b+c;
a = b+c;
```

- specifying a function – an example of the definition and use of a function is given below. This function determines the greater of two numbers. After introducing this function into the model, the modeller can use it anywhere on the RHS of a definition, as, for example, in the definition: "`a is greater(b, c);`".

```
func greater {
    para x, y;
    if (x>y) result = x;
    else result = y;
    return result;
}
```

- specifying an action – an example of an action is given below. An action in Eden is a 'triggered procedure'. In this case, the action `testaction` redefines `a`

whenever the variable `m` changes. The current definition of `a` depends on the value of `m`.

```
proc testaction: m {
    if (m>0) a is b+c;
    else a is b*c;
}
```

- basic means for querying a definition – examples of two different types of query are given below. The first statement asks for the current definition of variable `x`. The second statement returns the current value of `x`. In both cases, the output is directed to the 'output window'.

```
?x;
writeln(x);
```

Donald

Donald is a definitive notation for 2D line-drawing. Its design was first conceived by David Angier in his final year undergraduate project, and later extended by Beynon et al. [Bey86b]. Donald supports a variety of data types including `integer`, `real`, `char`, `point`, `line`, `shape`, `arc`, `circle`, `ellipse`, `rectangle` and `label`. Drawings are grouped into `viewports`. Each `viewport` is like a drawing board with independent coordinates. Variables in Donald need to be declared before their use. There is no procedural assignment in Donald. In Donald, the equals sign '`=`' denotes a definition, rather than a procedural assignment as in Eden. Each statement in Donald is terminated by a carriage-return. For example, Listing 2.1 below shows sample Donald definitions for specifying a circle within a square on the screen (see Figure 2.8 for the result in a screen capture).

```
1.   %donald

2.   viewport testDrawing

3.   rectangle theRectangle
4.   circle theCircle
5.   int x1, y1, x2, y2, centreX, centreY, radius

6.   x1 = 100
7.   y1 = 700
8.   x2 = 300
9.   y2 = 900
10.  centreX = x1 + (x2 - x1) div 2
11.  centreY = y1 + (y2 - y1) div 2
12.  radius = (x2 - x1) div 2

13.  theRectangle = rectangle({x1,y1},{x2,y2})
14.  theCircle = circle ({centreX,centreY},radius)
```

Listing 2.1: Definitions for drawing a circle within a square in Donald

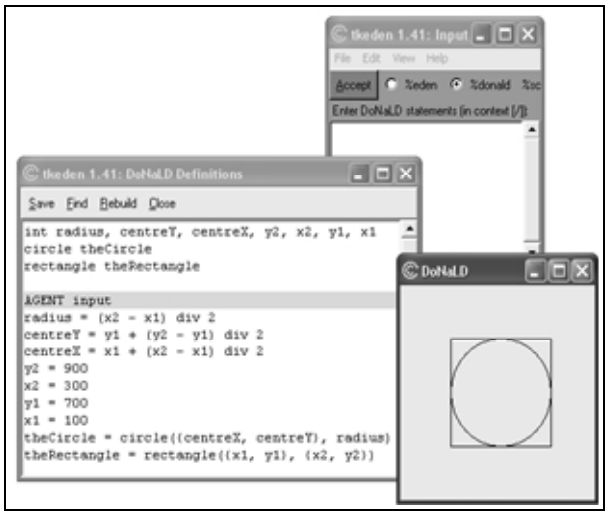

Figure 2.8: Screen capture of a sample Donald drawing

Scout

Scout (SCreen layOUT) is a definitive notation designed for specifying the contents and layout of windows on screen. It was introduced and implemented by Y. P. Yung in 1992 [Yun92]. The basic data types include `integer`, `string`, `point`, `box`, `frame`, `window` and `display`. In a Scout script, as in Donald, variables need to be declared before use, and the equals sign '`=`' denotes a definition. Each statement is terminated by a semi-colon '`;`'. Listing 2.2 shows sample Scout definitions that specify a window containing the string "move me!" that can be moved about the screen by a drag-and-drop mouse operation.

```
1.   %scout

2.   display d;
3.   window w;
4.   integer x,y;

5.   w={
6.     frame: ([{x,y},{x+60,y+40}])
7.     type: TEXT
8.     string: "move me!"
9.     sensitive: ON
10.    bgcolor: "blue"
11.    fgcolor: "white"
12.  };

13.  d=<w>;
14.  screen= d;
```

```
15.  %eden

16.  proc dragdrop: w_mouse_1{

17.  /* button pressing */
18.  if(w_mouse_1[2] == 4){
19.     logicalx=w_mouse_1[4];
20.     logicaly=w_mouse_1[5];
21.  }

22.  /* button released */
23.  if(w_mouse_1[2] == 5){
24.     x=x+w_mouse_1[4]-logicalx;
25.     y=y+w_mouse_1[5]-logicaly;
26.  }

27.  }
```

Listing 2.2: Definitions for specifying a movable window in Scout with a supporting Eden action

Lines 2-4 contain some Scout variable declarations. Lines 5-12 define the window. Lines 13-14 put the window on the screen. Lines 15-27 is an Eden action that recalculates the coordinates of the window whenever the modeller does a drag-and-drop mouse operation. The action is triggered by the variable `w_mouse_1` (line 16). Because the window `w` is defined to be sensitive (line 9), redefinitions of this variable are generated by mouse events. The variable `w_mouse_1` records the current status of the mouse and cursor position using an Eden list. A screen capture of the result of inputting the definitions and action in Listing 2.2 is shown in Figure 2.9 below.

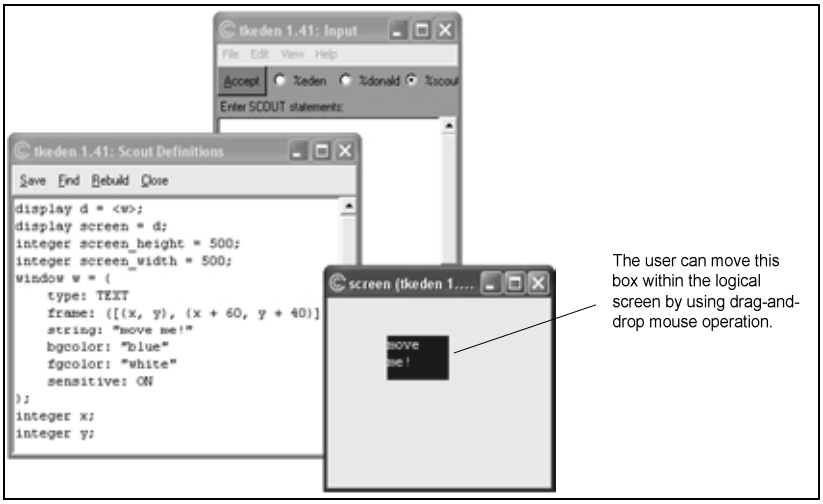

Figure 2.9: Screen capture to illustrate the use of Scout

The way in which Eden, Donald and Scout are integrated in TkEden is depicted in Figure 2.10. The core of TkEden is the Eden interpreter that is responsible for automatic dependency maintenance and for generating output. The Donald and Scout scripts specified by the modeller are translated to Eden script and actions to be

interpreted by the Eden interpreter. Other definitive notations can also be integrated into TkEden in a similar way. This is in keeping with Eden's primary role as an evaluator for definitive notations.

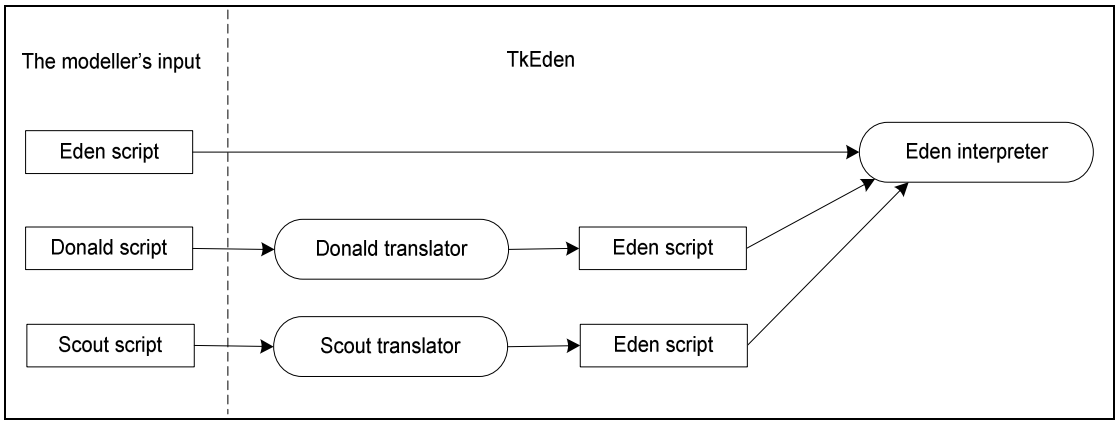

Figure 2.10: The integration between Eden, Donald and Scout in TkEden

TkEden is the principal tool for EM, and has now been used by several hundred students at Warwick University. A few hundred models have been built by using TkEden and its variants and many of these are available to download from the web [EMWeb]. Most of them include descriptions and tutorials that help users to learn about EM. Throughout this thesis, we shall use TkEden to build illustrative examples. In addition, in chapter 7, we evaluate TkEden as a tool to support modelling activities in the DMF. The prospects for introducing new EM tools will also be discussed in both chapters 7 and 8.

2.3.2 The Language for Specification and Description (LSD)

The LSD (Language for Specification and Description) notation was originally developed by Beynon in collaboration with Norris of British Telecom in 1986 to support a special form of agent-oriented analysis for concurrent systems [Bey86a, Bey88c]. Subsequently, Slade [Sla90] developed the design of LSD and investigated the scope for using LSD to generate executable models. The key activities that are performed in an LSD analysis are *identifying agents* and *classifying the observables that are deemed to govern their interaction*.

An LSD account of an agent classifies the observables associated with it into five categories, namely *state*, *oracle*, *handle*, *derivate* and *protocol*. Their meanings are briefly explained as follows:

- state – contains observables that the agent owns. The existence of these observables is dependent on the existence of the agent.
- oracle – contains observables that the agent may respond to.
- handle – contains observables that are conditionally under the agent's control.
- derivate – contains definitions that specify indivisible stimulus-response relationship between observables.
- protocol – contains possible actions that the agent can perform subject to certain enabling conditions being met.

For an illustrative example, consider the description of the Vehicle and Driver agents from a vehicle cruise control simulation in Listing 2.3 [Bey92a, Adz99]. Note that an observable does not necessarily have a single exclusive classification. For example, `engineStts` is both an oracle and a handle for the `Driver` agent. Also, because the same observables can be associated with different agents, an LSD account helps to express the subjective views of the agents which are not explicitly modelled in an ISM.

```
agent Vehicle{

  state:
    mass     /*total mass of car*/
    actSpeed /*actual speed*/
    accel    /*acceleration*/
    windF    /*wind resistance force*/
    brakF    /*braking resistance force*/
    ...

  oracle:
    brakePos /*brake position*/

  derivate:
    windF = windK * sq(actSpeed)
    brakF = brakK * actSpeed * brakPos
    accel = (traceF-brakF-windF)/mass
    actSpeed = integ_wrt_time(accel,0)
    ...
}
```

```
agent Driver{

  oracle:
    engineStts /*engine settings*/
    cruiseStts /*cruise settings*/
    ...

  handle:
    brakePos
    engineStts
    cruiseStts
    ...

  derivate:
    brakePos = user_input(brakPos_Type)

  protocol:
    (engineStts == off)-> engineStts = on
    (cruiseStts != off)-> cruiseStts = off
    ...
}
```

Listing 2.3: Fragments of the Vehicle Cruise Control Simulation in LSD notation

There is no prescribed role for LSD analysis within the DMF because in general we can perform modelling activities without explicitly thinking about the concepts of LSD. However, LSD provides a means to classify observables that can be used in a variety of ways in connection with the construction of ISMs. An initial LSD analysis has been the basis of several ambitious EM models (such as the five-a-side football simulation [Tur00] and the train arrival and departure model [Bey90b]), but modellers have the freedom to derive their own ways of analysis according to context. For this purpose, the LSD notation can be interpreted and used in three ways: *for construal*,

for description and *for specification*.

For construal. An LSD analysis can be used to support the activity of constructing an ISM. In this context, LSD notation is used to frame an account that expresses the way in which the modeller construes the phenomenon of current interest. There are two aspects to this construal: the identification of agents as groups of observables ('agentification'), and the exploration of the role that various observables play in mediating their interaction, as typically disclosed through projected or actual role-playing activity. For example, by thinking about the interaction between the driver and the vehicle, we are led to classify `brakePos` as an oracle observable for the `Vehicle` agent but a handle for the `Driver` agent. An LSD account of a phenomenon helps the modeller to acquire and document evolving understanding of the phenomenon that can be provisional, subjective and personal. It typically helps to stimulate a rich set of questions concerned with the phenomenon that would be difficult to identify without thinking about agents and their access privileges and interactions.

For description. The LSD notation can be used to provide information about the current status of an ISM. For instance, it can be very hard to know how to interact with a complex ISM. We can use LSD to document the intended user interactions, or – more generally – experimental interactions representative of the most interesting scenarios known to the modeller. In this way, an LSD description serves a useful role in communication between modellers as a complement to the sharing of an ISM.

For specification. The LSD notation can be used as a specification language to circumscribe understandings. However, the LSD notation has no formal operational semantics, and an LSD specification cannot be directly or automatically translated into an executable model (for more discussion of the issues involved the reader can refer to [Bey88a, Bey90a]). To give an operational interpretation to an LSD account additional assumptions have to be introduced.

2.4 Summary

In this chapter, we have identified the roots of EM in a commonsense way of understanding phenomena. EM is associated with the conception of concurrency as experienced in our everyday life in terms of observation, agency and dependency.

This philosophy leads to the Definitive Modelling Framework (DMF), which is characterised by an interactive style of model building based on a distinctive mode of representation and style of modelling activity. The DMF represents agents using groups of definitions and actions. Modelling activities in the DMF include agentification, role-playing and collaborative definitive modelling. The DMF supports an interactive style of model building, in which the use of the computer resembles the use of an instrument. We have also introduced the TkEden tool and its associated definitive notations that are used in practical EM, and described the role of LSD in EM.