Chapter 2

Background to EM, PD and SD

This thesis draws extensively on the concepts and principles of Empirical Modelling, product design and software development. The principal aim of this thesis is to investigate the suitability of Empirical Modelling as a framework for software development. Product design provides a context to this investigation to ensure that both the creative and analytical aspects of Empirical modelling are considered.

This chapter introduces Empirical Modelling, product design and software development. It presents the basic concepts and principles used throughout the thesis. Empirical Modelling is introduced as a situated computer-based approach to modelling developed at the University of Warwick by the Empirical Modelling Group; the account of product design is based on Pugh's concept of total design [Pug91], with emphasis on the conceptual design phase; a mainstream view of software development is adopted, focusing on object-oriented analysis.

2.1 Empirical Modelling

Empirical Modelling (EM) is a new approach to computer-based modelling that has emerged in the last few years during research by the Empirical Modelling Group at the University of Warwick. The choice of epithet *empirical* reflects the fact that our approach is rooted in observation and experiment. The emphasis on modelling what is experienced rather than preconceived distinguishes our approach from traditional approaches to computer-based modelling.

This section draws on the experiences of modelling an OXO game to illustrate EM. For further details of modelling a game of OXO the reader is invited to read the paper [BJ94] which is an elaborated account for publication. The reader may also wish to refer to the account of modelling a sailboat in Appendix B whilst reading this section.

2.1.1 General concepts and principles

Although EM is quite a recent innovation it is the result of long-term research into a particular kind of modelling called *agent-oriented modelling over the definitive representation of state* (AOMDRS). The concepts and principles specific to EM will be introduced later in this section. Before doing so the main concepts and principles underlying AOMDRS will be described.

There are three key concepts in AOMDRS: observable, agent and dependency. These concepts are defined as follows:

- an **observable** is any feature of the subject or model that can be reliably perceived, identified, and compared with similar features [Rus97];
- an **agent** is anything (human or otherwise) capable of changing the state of the subject or model [Rus97];
- a **dependency** is a relation between observables such that changing the value of a certain observable has a predictable effect upon the values of other observables.

The key notions of observable, agent and dependency are interrelated, thus providing a cohesive conceptual framework to AOMDRS: an observable has meaning with reference to the perception and interaction with a feature of the subject or model by an agent; a state has meaning with reference to simultaneous observations made by an agent [BRY90]; the simultaneous observation of observables gives the concept of dependency and state its meaning; action by agents is mediated through dependencies between observables [Bey97].

The central principle of AOMDRS is to establish a correspondence between two sets of observables: the real-world observables that represent the subject and the reference set of observables that is defined by the features of the model. Establishing the correspondence is an iterative process, leading to successive refinement of the model and circumscription of the subject, that involves observation and experiment within the model and situation. The objective in setting up this correspondence is to achieve consistency between the way in which sets of observables are indivisibly linked in change in the subject and the way in which the corresponding sets of observables are indivisibly linked in change in the model [BNR95].

The correspondence between subject and model is achieved in AOMDRS by the modeller using the complementary techniques of observation and agent-oriented analysis and definitive representation of state:

- **Observation and agent-oriented analysis** involves the identification of observables, agents and dependencies in the subject, as shown in Example 2.1. This is achieved through interaction by the modeller with the subject and model in parallel. Through interaction the modeller reconciles what they observe of the subject with their beliefs about the subject as represented in the computer model. The modeller's beliefs about the subject are also recorded as a document in an LSD specification [Bey86b].

  An LSD specification details the observables whose values can act as stimuli for an agent (its oracles), that can be redefined by the agent in its responses (its handles), those observables whose existence is intrinsically associated with the agent (its states) and those indivisible relationships between observables that are characteristic of the interface between the agent and its environment (its derivates). The repertoire of possible state-changing actions of agents is also recorded (its protocol) [Bey97].

  An LSD specification generally admits many different operational interpretations, corresponding to different presumptions about the environment in which agents interact, and the nature and reliability of their stimulus-response patterns. The LSD specification can be given an operational meaning within the framework of the Abstract Definitive Machine (ADM) [Sla90]. In the ADM, transitions are represented by parallel redefinition of variables in a definitive script. In this context, the user can interact freely with the model as a superagent both to introduce new redefinitions on-the-fly and to dictate the pattern of agent-interaction.

Example 2.1. Observation and agent-oriented analysis in OXO. Four agents can be identified by the modeller in the game of OXO:

- the board;
- the player who places Xs on the board;
- the player who places Os on the board;
- the umpire who decides who plays next and who has won.

The player agents are the easiest to identify because these are the roles played by the modeller in games of OXO. The player interacts with the board during play even though the status of the board as an agent is somewhat obscure (Example 2.7). The decision by the modeller to identify an umpire agent indicates that the modeller views playing and the rules of play as conceptually distinct.

The modeller defines each of the agents in LSD. The LSD definitions of the board, player and umpire agents could be as follows:

```
agent board() { state Board = BlankBoard }

agent player(P, O) {
state     choice
oracle    turn   Board
handle    Board
protocol
  turn == P && available(Board, choice) -> take(Board, choice),
  !available(Board, choice) -> make_new_choice()
}

agent umpire() {
state     turn
oracle    numofX    numofO
handle    Board
derivate
  turn = (numofX > numofO) ? player_X : player_O
protocol
  win(player_X) -> congratulate(player_X); Board = BlankBoard
  win(player_O) -> congratulate(player_O); Board = BlankBoard
  tie() -> declare_tie(); Board = BlankBoard
}
```

These agent definitions record the observations and relationships between observations identified by the modeller whilst playing OXO. The definitions are personal (what is involved in making a new choice?) and subject to revision (the umpire needs to know who played first in order to determine whose turn it is when there are an equal number of noughts and crosses on the board).

- **Definitive representation of state** is the the process whereby the modeller represents observables by variables and introduces definitions (similar in character to the defining of formulae for cells of a spreadsheet) to express the way in which the values of observables in the subject are interdependent, as shown in Example 2.2. Such a set of definitions - a definitive script - represents a possible experimental observation, and redefining a variable corresponds to changing an experimental parameter [ABCY94c].

  A language that may be used to write a definitive script is referred to as a definitive notation. Each definitive notation is conceived with a mode of visualization in mind. Each has its own set of data types and underlying algebra, appropriately chosen for the scope of the application. ARCA, DoNaLD and SCOUT are examples. ARCA [Bey86a] was designed for the display and manipulation of combinatorial diagrams, DoNaLD (Definitive Notation for Line Drawing) [ABH86] for two-dimensional line drawing and SCOUT [Dep92] for describing screen layouts. To complement these special purpose notations, the definitive language EDEN [YY88, Yun90] incorporates C-like data types and operators to facilitate more general applications and the implementation of other definitive notations [BYCH92].

Figure 2.1 shows the relationship between the tools and notations used in observation and agent-oriented analysis and the definitive representation of state. The tkeden interpreter, used to implement the ADM, is discussed in Section 2.1.4.

AOMDRS is based upon the concepts of observable, dependency and agent, and the principles of analyzing observables and agents and the definitive representation of state as described above. However, these descriptions are unlikely to satisfy those who are looking for a systematic approach to the analysis and precise representation of systems. Most would probably accept that the concepts and principles have an intuitive meaning and that they can be interpreted in many different ways. In particular, there is no formal definition of observable, dependency and agent. There is no systematic way of identifying these in the subject either.

Example 2.2. Definitive representation of state in OXO. In parallel with the definition of LSD agents the modeller can translate the partial agent definitions into ADM entities and can implement the entities as definitive scripts. The translation essentially replaces derivates by definitions and protocols by actions. This process requires the modeller to address issues of synchronization between observables that emerge when the LSD specification is interpreted operationally.

The model of the OXO game is developed incrementally through a sequence of modelling steps, each of which leads to the construction of a definitive script, named as an EDEN file, capturing assumptions about the synchronization of observations as identified during agent and observation-oriented analysis (Example 2.1):

- What is the geometry of the board ? (`geometry.e`)
- How does the actual board and what I perceive conform? (`display.e`)
- Can I interpret the board in OXO terms? (`status.e`)
- What considerations guide me in contemplating the next move? (`sqvals.e`)
- Whose turn is it to play? (`gamestate.e`)

This hierarchical organization of the modelling reflects the hierarchy of perceptions and actions underlying OXO-playing, ranging from low-level capabilities to see the board and apprehend geometric patterns to high-level abilities to interpret positions and apply rules [BJ94].

The final stage of model construction involves automating one of the players in the game of OXO. The ADM player entity translates into an EDEN triggered action (`control.e`) that takes a turn at OXO.

Each of the files of definitions correspond to the modeller's answer to the questions posed by observations and agency identified in the LSD specification. For example, the file `geometry.e` contains the following EDEN definitions:

```
allsquares is [s1,s2,s3,s4,s5,s6,s7,s8,s9]
lin1 is [s1,s2,s3]
lin2 is [s4,s5,s6]
...
linesthru1 is [lin1,lin4,lin7]
linesthru2 is [lin1,lin5]
...
linesthru is [linesthru1, linesthru2, ..., linesthru9]
```

These definitions indicate the decision by the modeller to represent the board geometrically in terms of lines of squares. This corresponds to how the modeller perceives the board whilst playing the game of OXO.

The original aim of this thesis - to investigate the suitability of our approach to modelling as a SD method - made it necessary to find a more formal definition of AOMDRS. The search for a more formal definition and the issues that emerged during the search contributed to the emergence of EM. EM has AOMDRS as its basis with additional concepts and principles that give EM integrity and distinguish it from other approaches to computer-based modelling. The uniqueness of our approach to computer-based modelling results from the emphasis on modelling what is experienced rather than what is preconceived.

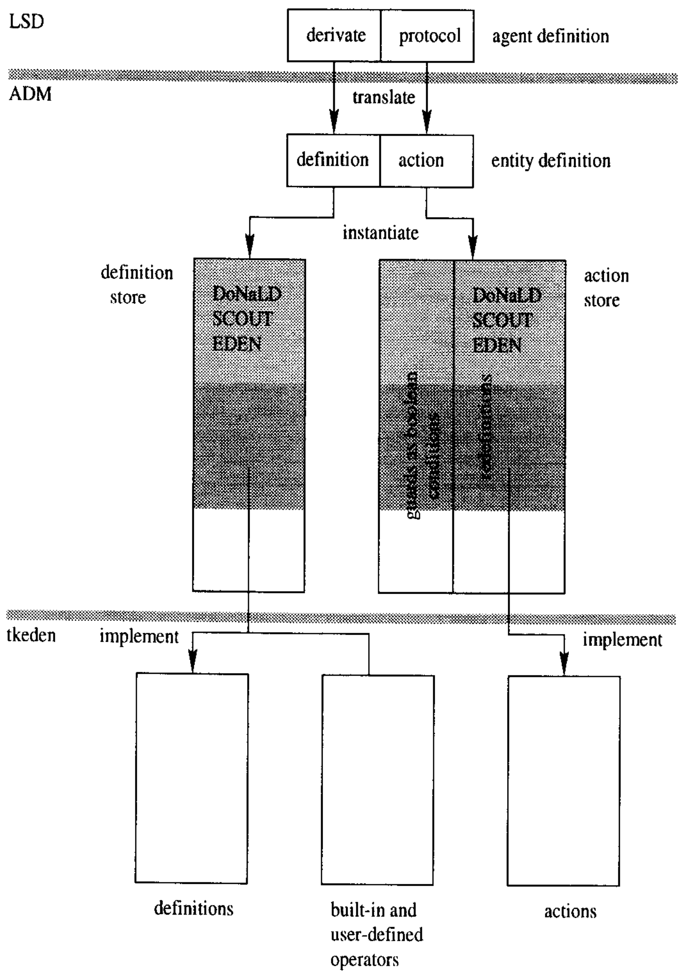


Figure 2.1: Tools and notations in EM.

2.1.2 Agentless systems

The original aim of this thesis was motivated by the apparent similarity between our approach to modelling and the Shlaer-Mellor method of object-oriented analysis

(SMOOA) and design (SMOOD) [SM88, SM92] as was being used at the IBM WSDL (Appendix B):

- The definitive scripts combined with a formal definition of the ADM provide a mathematical definition of the behaviour of the model (albeit subject to assumptions about interaction from the environment). This behaviour could, in principle, be recreated using other programming languages, such as C++ as used at the IBM WSDL.
- The LSD specification could be viewed as being a descriptive version of the Entity Relationship Diagram (ERD) used in SMOOA.
- The concepts of observable, agent and dependency correspond to concepts in SMOOA: observable to variable; agent to object; dependency to relationship between objects.
- Reconstructions of modelling processes tend to follow a pattern (LSD specification, visualization followed by animation) corresponding to SMOOA, SMOOD followed by coding.

However, it has since been recognized that these similarities are associated with a particular view of our modelling approach characterized by the absence of any consideration of the role of the modeller in the process. This absence is a direct consequence of focusing on existing models and reconstructions of the modelling process.

A view of EM that ignores the agency of the modeller effectively reduces it to what is termed an agentless or 0-agent system in EM, as shown in Example 2.3. In the absence of the modeller to explain what is meant by their model it is necessary for the model to use established, typically linguistic, conventions for representation. Certainly most LSD specifications, definitive scripts and accounts of modelling are written in such a way that they can be understood by those familiar with EM. These conventions must not be susceptible to change by individual agents. In general, a 0-agent system is one that, at some level of abstraction, does not exhibit change.

Example 2.3. 0-agent view in OXO. The modelling of the OXO game can be characterized as follows by ignoring the agency of the modeller and focusing instead on the artefacts produced and reconstructions of the modelling process:

- The definitions and actions of the board, player and umpire entities have an unambiguous behavioural interpretation within the framework of the ADM that could be implemented using conventional programming languages.
- The LSD board, player and umpire agents appear to specify the corresponding ADM entities.
- The observables, derivates and protocols of the board, player and umpire agents are given formal definitions in terms of the variables, definitions and actions of the corresponding ADM entities.
- Construction of the model follows an order (`geometry.e`, `display.e`, `status.e`, `sqvals.e`, `gamestate.e`, `control.e`).

Such a view of the modelling process gives it the character of a SD method. In fact, the definitive scripts generated during the modelling of the OXO game were used to implement a Pascal version of the OXO game.

2.1.3 Single-agent systems and modelling

Although the 0-agent view of EM is a valid interpretation it does not capture the essence of EM as experienced by the modeller. The use of the epithet *empirical* is meant to convey the central importance of the modeller's agency during the modelling process:

- The modeller correlates the experiences of the subject and the computer model through observation and experiment: the meaning of the computer model is defined by interaction [Bey97].
- The LSD specification records the observables, dependencies and agency as perceived by the modeller in the subject and represented in the computer model.
- The meaning of observables, agents and dependencies is given by the interaction of the modeller with the subject and model.

- The modeller is a free agent with actions determined by what the modeller knows, perceives and expects.

The activity of EM is an archetypal example of what is termed a 1-agent system in EM. Modelling involves the modeller changing the state of the model and subject by interacting with observables with a view to establishing a correspondence between the states, as shown in Example 2.4. In these cases the modeller is the sole instigator of state change within the modelling process. In general, a 1-agent system is a system in which one agent has the capacity to surprise.

Example 2.4. 1-agent view in OXO. By considering the experience of the modeller during the modelling of the game of OXO an altogether different view of the modelling process from that in Example 2.3 emerges. This 1-agent view of the modelling process is characterized by the following:

- The correspondence between the game of OXO and the computer model may change in unpredictable and surprising ways through interaction by the modeller, such as altering the shape of the board or the rules of play.
- The LSD description records the characteristic observables and dependencies of the board, player and umpire as identified by the modeller, such as the player seeing the board as lines and the umpire judging the next move based on the number of noughts and crosses.
- The meaning of observables and dependencies emerges through playing OXO and interacting with the model. For example, the observable `linesthru` is derived from the experiences of the modeller playing and representing the game of OXO.
- The order of model construction (`geometry.e`, `display.e`, `status.e`, `sqvals.e`, `gamestate.e`, `control.e`) reflects the way the modeller conceives the game of OXO.

This view of modelling the game of OXO captures the essence of EM. EM is used by the modeller to support their conceptualization of the game of OXO rather than as a method to represent preconceptions about the game.

This 1-agent approach to modelling is significant because models need only be understood by the modeller. In EM the objective is for the modeller to acquire an understanding of the subject themselves. This understanding does not require models that are understood by others, only models that can be seen as corresponding to the subject by the modeller. Such models are subjective and personal in nature involving conventions whose meanings are dependent upon the modeller. A secondary objective in EM is typically to represent these more objectively using established conventions for representation so that they emerge within the 0-agent view of EM.

2.1.4 Computer as artefact

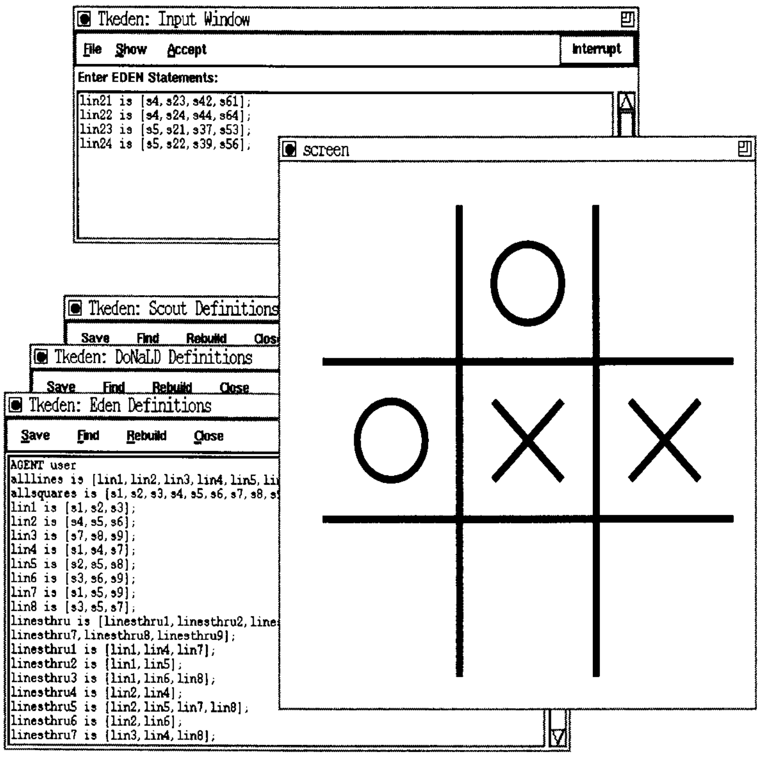
It is the unusual status given to the computer in EM that allows the 1-agent approach to modelling. In EM the computer is only significant in so far as it serves as a physical instrument with which the modeller interacts. This is in contrast to the way in which the computer is conventionally regarded in classical computer science as a means to implement an abstract algorithm or computation. In effect, it is how the user perceives the computer as a physical object that matters in EM, not the invisible mechanism by which this object is specified [BNR95].

The status of the computer model in EM is similar to that of the spreadsheet. The only changes to the state of a spreadsheet are via actions on the part of the user. However, the essential spirit of EM is better represented where there is an explicit experiential aspect to the model. This could be achieved by the visualization of spreadsheet data. In EM the variables that appear in definitive scripts typically have an experiential significance - they may refer directly to entities visible to the computer user, such as points, lines, geometric attributes or windows on the screen for instance.

The EM tool that gives the computer the special quality that supports 1-agent modelling is the **tkeden** interpreter (Example 2.5). Almost all the models that have been developed using EM principles have been represented using the **tkeden** interpreter. This applies even to those that are constructed using the ADM, since this is at present implemented via a translator that acts as a front-end to **tkeden**, as shown in Figure 2.1.

A typical **tkeden** file comprises three kinds of construct: definitions, functions and actions. Definitions are formulated in terms of variables that represent scalar quantities, text strings and recursive non-homogeneous lists, as well as visually significant elements such as points, lines, and shapes in the form of planar line drawings, and windows in the screen layout. Functions serve as user-defined operators on the RHS of definitions; these supplement standard built-in operators that are used to define scalar, structural and geometric relations. Actions are specified as procedures that are triggered by changes to the values of particular variables [Bey97].

Example 2.5. Computer as artefact in OXO. Figure 2.2 shows a screen-shot of the EM **tkeden** during the modelling of the OXO game.



```
lin21 is [s4, s23, s42, s61];
lin22 is [s4, s24, s44, s64];
lin23 is [s5, s21, s37, s53];
lin24 is [s5, s22, s39, s56];
```

```
AGENT user
alllines is [lin1, lin2, lin3, lin4, lin5, li
allsquares is [s1, s2, s3, s4, s5, s6, s7, s8, s
lin1 is [s1, s2, s3];
lin2 is [s4, s5, s6];
lin3 is [s7, s8, s9];
lin4 is [s1, s4, s7];
lin5 is [s2, s5, s8];
lin6 is [s3, s6, s9];
lin7 is [s1, s5, s9];
lin8 is [s3, s5, s7];
linesthru is [linesthru1, linesthru2, line
linesthru7, linesthru8, linesthru9];
linesthru1 is [lin1, lin4, lin7];
linesthru2 is [lin1, lin5];
linesthru3 is [lin1, lin6, lin8];
linesthru4 is [lin2, lin4];
linesthru5 is [lin2, lin5, lin7, lin8];
linesthru6 is [lin2, lin6];
linesthru7 is [lin3, lin4, lin8];
```

Figure 2.2: **tkeden** being used to model OXO game.

There are separate windows showing the EDEN, DoNaLD and SCOUT definitions. There is a window that shows the current state of the visualization. The **tkeden** input window is used to enter redefinitions of the scripts on-the-fly (the redefinitions shown are some of those being entered by the modeller to change the geometry of the board for 3-dimensional OXO).

The experiential character of 1-agent models in EM has crucial significance in respect to the relationship between computer model and subject. The focus on observables and dependencies in EM, when combined with mechanisms that make these directly perceptible in interaction, is the means of ensuring that what is observed of the artefact (albeit as in caricature rather than as in realism) conforms to what is observed in the subject [Bey97].

2.1.5 Multi-agent systems and modelling

Although EM is essentially about the modelling of a subject by a single modeller using a computer it is often within the context of other human participants, as discussed in Example 2.6. The advantage of 1-agent modelling is that the modeller need not concern himself with how his model is understood by others. However, for most applications there comes a stage in modelling when the modeller has to communicate his use of artefacts and perhaps justify his actions to others. In general, a system with more than one agent (human or otherwise) is termed an n-agent system in EM.

By considering EM as a group activity it is possible to identify and associate roles with the human agents involved. The following are typical of the roles observed during modelling:

- the modeller who constructs the computer model;
- those familiar with the subject domain or who have a stake in the modelling;
- the objective or external observer whose view of the subject combines those of other agents.

Each of these roles might be fulfilled by different people in EM or a single person might have a number of roles. When more than one person is involved in EM communication becomes an important issue. In 1-agent EM the modeller typically assumes multiple roles in which communication is not an issue.

The EM concept of n-agent systems extends beyond social systems to systems including non-human agents. By associating human-like characteristics with the concept of agent the modeller assumes the roles of inanimate objects. This conceptual mechanism for assisting in the modeller projecting their own general characteristics onto the subject being modelled is an essential principle of EM.

Example 2.6. n-agent view in OXO. As was mentioned in Example 2.1 the modeller identified four agents in the game of OXO:

- the board;
- the player who places Xs on the board;
- the player who places Os on the board;
- the umpire who decides who plays next and who has won.

During the modelling of the game of OXO the modeller assumes the roles of the agents. In this way 1-agent modelling can be used to model n-agent systems.

The modeller adopts two different perspectives on the game of OXO during modelling:

- the game of OXO from the viewpoint of the board, player and umpire agents, and
- the game of OXO from the viewpoint of an external observer who sees the corporate effect of the board, player and umpire agents interacting.

The individual viewpoints are specified in LSD and the external viewpoint is represented by animating the agents within the ADM framework.

There comes a stage in modelling the OXO game when the modeller will want to test his model by letting others interact with it. By letting others interact with the model the modeller is testing his beliefs about the game of OXO and giving integrity to the model. This transforms the modelling of the OXO game from 1-agent to n-agent.

There are essentially two different ways in which EM can be applied to modelling systems of multiple agents [Bey97]:

- **Scenario 1** The modelling activity is centred around an external observer who can examine the system behaviour, but has to identify the components agents and infer or construct profiles for their interaction;
- **Scenario 2** The system can be observed from the perspective of its component agents, but an objective viewpoint or mode of observation to account for the corporate effect of their interaction has to be identified.

In many applications it is appropriate to consider both scenarios concurrently, with a view to reconciling global and local perspectives on the behaviour of a system.

It is in connection with systems with more than one agent or observer that LSD descriptions become significant. Modelling of n-agent systems can be viewed as involving two complementary principles that constitute concurrent engineering in EM [ABCY94c, ABCY94a, ABCY94b]:

- specifying agents in LSD by considering them in isolation;
- introducing a context for interaction by animating agents using ADM.

Arguably modelling n-agent systems is made difficult by the complex interaction between agents able to change the same observables. The modeller describes agents in LSD without having to address which observables are shared. Such issues are addressed in the ADM when the synchronization of interaction between entities is important.

2.1.6 Agent concept

The status of the modeller in 1-agent modelling is central to EM therefore the most appropriate way to conceive other agents is as having the same general characteristics of the modeller.

The characteristics of the modeller can vary, resulting in a broad interpretation of agency in EM. This broad spectrum of agency is categorized according to different views of an agent [Bey97], as illustrated in Example 2.7:

- **View 1** An entity comprising a group of observables with unexplored potential to affect system behaviour;
- **View 2** A View 1 agent that is capable of particular patterns of stimulus-response within the system.
- **View 3** A View 2 agent whose pattern of stimulus-response interaction can be entirely circumscribed and predicted.

In EM, each of these views has a different status, and there is a tendency to progress from the first to last view of an agent during modelling. EM is of interest somewhere between View 1, where the agent concept is vacuously broad, and View 3, where it is impotent. This interest centres around our uncertainty about the status of entities in respect to agency. In View 1 our concern is whether an entity has any influence and in View 3 our concern is whether the exact nature of its influence is known.

The classification of agency according to these views is not a formal matter. Agency is being invoked as a conceptual device fundamentally associated with how phenomena are construed to occur. EM promotes the view that agency is only meaningful in relation to the development of understanding.

Example 2.7. Classification of agents in OXO. The agents in the OXO game and its representation show the full range of agency accommodated in EM:

- The opponent at the start of modelling and the board are examples of a View 1 agent with unexplored potential for affecting the game of OXO.
- The opponent, once the player-modeller has identified it as having a similar role to themselves, is an example of a View 2 agent with identified patterns of stimulus-response in the game of OXO.
- An automatic player entity is an example of a View 3 agent whose pattern of stimulus-response is made entirely predictable by the framework of the ADM.

Although all these views of agent appear within the modelling process the focus of attention is on the View 2 player agents. EM involves the modeller moving from a View 1 through to a View 3 of agents.

2.1.7 Conceptualization

EM can be thought of as the means by which the modeller represents the conception of the subject as it evolves [Bey97]:

1. Interaction with artefacts: identification of persistent features and contexts.
2. Practical knowledge: correlations between artefacts, acquisition of skills.
3. Identification of dependencies and postulation of independent agency.
4. Identification of generic patterns of interaction and stimulus-response mechanisms.
5. Non-verbal communication through interaction in a common environment.

6. Phenomenological uses of language.

7. Identification of common experience and objective knowledge.

8. Symbolic representations and formal languages: public conventions for interpretation.

The stages of EM represent a progression from a subjective to an objective view of the subject, as in Example 2.8. The early stages correspond to the modeller's view of the subject during 1-agent modelling. In later stages the modeller develops methods of communication as typified in n-agent modelling. Finally the model acquires a meaning independent of the subject and modeller (0-agent system).

Example 2.8. Conceptualization in OXO. It is possible to match the stages in constructing the model of the OXO game given in Example 2.2 with the stages of conceptualization given in Section 2.1.7:

- Construction of `geometry.e` (What is the geometry of the board?) and `display.e` (How does the actual board and what I perceive conform?) corresponds to interaction with artefacts: identification of persistent features and contexts.

- Construction of `status.e` (Can I interpret the board in OXO terms?) and `sqvals.e` (What considerations guide me in contemplating the next move?) corresponds to practical knowledge: correlations between artefacts, acquisition of skills.

- Construction of `gamestate.e` (Whose turn is it to play?) corresponds to
  - identification of dependencies and postulation of independent agency;
  - identification of generic patterns of interaction and stimulus-response mechanisms;
  - non-verbal communication through interaction in a common environment;
  - situated use of language;
  - identification of common experience and objective knowledge.

- Construction of `control.e` corresponds to symbolic representations and formal languages: public conventions for interpretation.

The construction of the model of the OXO game reflects the conceptualization of the OXO game by the modeller.

It is this natural progression from the subjective to objective view of a system that provides the EM activity with its order rather than the modeller following a prescribed method. The aim of EM is to support the natural process of conceptualization rather than prescribe essentially unnatural methods of analysis.

2.1.8 Situating EM

The closest conventional computing comes to EM is in the use of spreadsheets. Using a spreadsheet illustrates agency in a 1-agent system. The semantically interesting state is in the relationship between the states of the spreadsheet and the part of the real-world it models. The only significant changes to the state are via actions on the part of the user. In [Nar93b] Nardi presents a particularly interesting study of the impact of spreadsheet use on the SD culture. The themes emerging from this study - support for interaction, re-use and extensibility - are consistent with our experience and aspirations for EM.

The distinction between EM approaches and formal approaches to describing behaviour in computer science is highlighted by Brian Cantwell Smith [Smi95, Smi87]. Smith distinguishes between the semantics of a program as it is understood in theoretical computer science and the relationship between these semantics and the external world. The real-world meaning of a program, for which EM provides a means of development [Bey92], is appropriately termed the "the semantics of the semantics" of programs by Smith. As Smith's analysis makes clear, knowing the semantics of a program and knowing how to deal with the semantics of the semantics of a program are quite different issues.

As the title "Empirical Modelling" suggests, our approach to modelling is rooted more in the philosophy of empiricism than rationalism and logic. The work of the American philosopher William James [Jam96] indicates that "Radical Empiricism", rather than traditional empiricism, provides the more appropriate philosophical foundation to our modelling method. James argues that by identifying sensory particulars the traditional empiricists break up the "conjunctive relations" that are "pure experience": "Conception disintegrates experience utterly" ([Jam96] p70), "[it] performs on conjunctive relations the usual rationalistic act of substitution -

[taking] them not as they are given in their first intention, as parts constitutive of experience's living flow, but only as they appear in retrospect, each fixed as a determinate object of conception, static, therefore, and contained within itself." ([Jam96] p236).

This philosophical outlook of James can be recognized in Gooding's account of scientific discovery and Faraday's discovery of electromagnetism. In his book [Goo90] Gooding rejects the conventional approach of analyzing workbooks and reconstructing methods. Instead, Gooding presents the notion of a "construal" that is "a means of interpreting unfamiliar experience and communicating one's trial interpretations" and uses this as a vehicle for understanding how scientists conceive and communicate their understanding of novel phenomena. There are parallels between the role of the construal in experimental sciences and the computer model in EM and the nature of the process of constructing such artefacts.

Understanding the behaviour of modellers inevitably leads to psychological considerations, in particular cognitive psychology because of the interest in understanding how modellers come to know about the subject during EM. As far as EM is concerned perhaps the most interesting explanations of cognitive processes are those that involve the creation of a model of the world in the mind, such as mental modelling proposed by Johnson-Laird [JL83]. This suggests that in constructing a computer model the modeller is mirroring and supporting mental processes. Other psychologists have tried to identify the qualities of models that support the conceptualization of novel systems [FWS92].

2.2 Product design

In this thesis PD means product design in the sense of Pugh's vision of *total design*: "the systematic activity necessary from the identification of the user need to the selling of the successful product to satisfy the need - an activity that encompasses product, process, people and organization" [Pug91]. His model of total design is meant as a framework for what design is rather than a prescriptive method of how design should be done. Pugh's view of design has been adopted as the basis of learning design in over 80 institutions within the United Kingdom. The books

"Total Design: Integrated Methods for Successful Product Engineering" [Pug91] and "Creating Innovative Products Using Total Design: the Living Legacy of Stuart Pugh" [Pug96] describe total design and are reviewed in Appendix D.

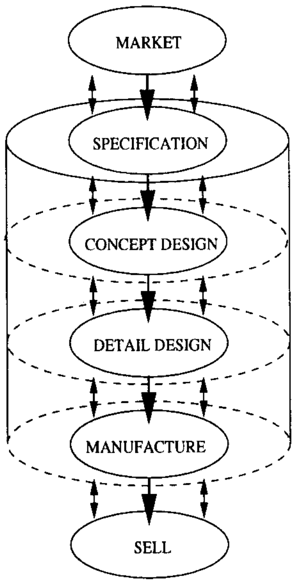


Figure 2.3: Activity model for total design.

The model of total design, shown in Figure 2.3, has a central core of activities all of which are imperative for any design irrespective of domain. The design core consists of the investigation of the market, development of a product design specification, conceptual design, detail design, manufacture and selling of the product. Design starts with a need that, when satisfied, results in a product that fits into an existing market or creates a market of its own. From the statement of the need a *product design specification (PDS)* is formulated which is the specification of the product to be designed. The PDS acts to constrain the total design activity by placing boundaries on the stages in the design core. Other constraints that are specific to particular designs, such as management, quality, information, techniques and technology, are discussed in Section 6.7 with respect to management and quality in EM and product design.

Although all the stages of total design are of equal importance the stages of specification, conceptual design and detail design are detailed below because they set the context for the discussion of design in this thesis:

- The starting point for any design activity is an investigation of the market from which a comprehensive PDS is prepared during the *specification* stage. Pugh makes it clear that the PDS is the specification of the *product to be designed*, not the specification of the product itself. The latter manifests itself only on completion of the design activity.

  At the end of the design activity the design of the product should fit the PDS that may have changed along the way. If during the design of a product there is good reason for changing the PDS then it is changed. It is considered by Pugh as an evolutionary, comprehensively written document which upon completion of the design activity has itself evolved to match the characteristics of the final product.

- The *conceptual design* stage is primarily concerned with the generation of solutions to meet the PDS. In fact, this stage combines the generation of solutions to meet the PDS with the evaluation of solutions to select the ones that best fit the PDS.

  In generating solutions the designer must come up with concepts which he believes fit the PDS and communicate these ideas for evaluation. Central to this conceptualization and representation is the process of synthesis. The designer mentally synthesizes familiar images and concepts from his knowledge and experience, with the PDS in mind, generating concepts for the system as a whole. The designer's ideas are represented as sketches, models, documents and prototypes for the purpose of evaluation.

  In evaluation choices have to be made about which solutions to reject and which solutions to keep for further refinement. Pugh argues that optimization provides a partial solution to the problem of evaluation because of its reliance on quantifiable evaluation criteria based on the PDS. Pugh suggests a total solution to the problem of evaluation based on decision matrices that he calls the method of *controlled convergence*: with concept names heading the rows and evaluation criteria heading the columns the design group comes to a consensus on scores to complete the matrix. It is expected that completing the matrix will lead to the emergence of new solutions to evaluate.

- Components and sub-systems are engineered in the *detail design* stage. During the conceptual stage of design the designer becomes increasingly involved in the detail design of the concept. The focus of design moves from the design as a whole to individual subsystems and components.

  By the end of the conceptual design stage the designer has detailed knowledge of the properties needed of components. This knowledge is stated in the form of a component design specification (CDS). The CDS is simpler than the PDS with a shift of emphasis. It is simpler because many of the criteria such as testing, packing, shipping, aesthetics and ergonomics are not relevant at component level. However, the performance, which is essentially the behaviour of the component, is important at the component level.

Pugh points out that, although the sequence of these stages is typically ordered as specification, conceptual design then detail design, the "design flow" is bidirectional between stages. Detail design can influence conceptual design which can in turn influence the specification of the product.

2.3 Software development

In this thesis SD means mainstream software development as exemplified by the Shlaer-Mellor object-oriented analysis and design method adopted at the IBM WSDL (Appendix B). Such approaches are characterized by an object-oriented analysis method for transforming the requirements for a system into code.

SD is traditionally divided into the stages of analysis, design, coding and testing or maintenance [Roy70, You92]. Originally meant to be in strict sequence [Roy70], more recent versions of the model [Boe85] show stages repeating and bidirectional flow between stages, with perhaps the most radical being the prototyping lifecycle [Boa84]. Although all of the stages are essential to SD this thesis concentrates on the stage of analysis. Analysis is the examination and representation of a real-world system for the purpose of designing and implementing software. The products of analysis typically determine what system is to be designed and implemented making it generally recognized as the most important stage in SD.

Most recently the discipline of requirements engineering [LK95, Lam88, Hof93, Poh96] has become associated with systems analysis [Gog94, SS96]. Requirements engineering is "the systematic process of developing requirements through an iterative co-operative process of analysing the problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained" [Poh96]. Requirements engineering results in the formulation of a precise description of the system known as a requirements specification or statement of requirements. Although it can take many forms, ranging from informal natural language to more formal graphical and mathematical notations [LK95], it is generally agreed [Lam88, Dav93, Hof93] that a statement of requirements should be, or aim to be, complete, correct, unambiguous, understandable, modifiable and consistent [DT90] with respect to the system.[1]

Throughout the brief history of SD there have been many methods proposed for performing analysis [DeM78, Jac83, SM88, CY90, Mar90, WBWW90, R^+91, Rum93, Jac92, SM92, Boo93, Boo86, YC75, Mey88]. Today, the most popular methods for analysis are object-oriented methods [You92] based on the notion of an Object. The object-oriented Object concept (starting with a capital to distinguish it from the word object that has a different meaning as discussed in Section 6.4.3) [Boo93, CY90, Mey88, Nie89] is a mechanism for abstraction and generalization. An Object consists of an interface and implementation. The interface is an abstraction of the implementation and the only part of the Object concept that is visible to clients of the Object. The implementation which provides the functionality defined in the interface is hidden. Objects with the same interfaces are classified together. The Object class definition contains the names of the service actions provided by the Object implementation. The class definition also defines the structure of the Object class in terms of other classes. In this way the Object concept deals with the representation, organization and abstraction of the structural, behavioural and func-

[1]This account is meant to represent the current status of requirements engineering. However, requirements is perhaps the most rapidly evolving field within SD. Section 6.5 discusses how the future of requirements engineering relates to EM.

tional aspects described in the statement of requirements. Object-oriented methods of analysis focus on the structure, behaviour and functionality of the system.

There are many methods of object-oriented analysis to choose from [Nie89, WBJ90, MP92b, FK92]. However, although each has different notations, their underlying concepts and principles are very similar. In this thesis a hybrid method is used that is based on the notations, concepts and principles of the established object-oriented analysis methods of Coad-Yourdon [CY90], Shlaer-Mellor [SM88, SM92, Lan93, FHRK93] and Rumbaugh [Jac92]. This hybrid is intended to highlight the essential nature of object-oriented approaches by stripping away the largely idiosyncratic stylistic complexities of specific notations [FS97, BRJ98b, BRJ98a].

Object-oriented analysis involves constructing separate models of the structure, behaviour and function of a system. The models are constructed in order, with the information given in each model being used in the construction of subsequent models. The models and order of development are as follows:

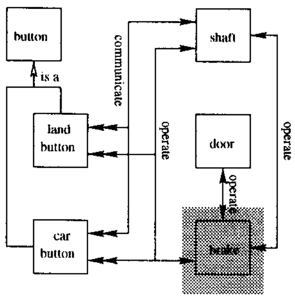
- The *structure model*, shown in Example 2.9, represents the organization of the system into Object classes. Labelled boxes represent Object classes. Labelled arrows between boxes represent structural, such as landing button *is a* button (inheritance), and functional associations between Object classes, such as shaft *operates* brake. Functional associations correspond to actions performed by Objects. Single and double headed arrows are used to indicate how instances of Object classes are associated with one another.

- The *behaviour model* or state model, shown in Example 2.10, represents each Object class lifecycles as a state-transition diagram. The states are represented by labelled boxes. The state transitions are represented as directed arrows, each labelled with an action and event name. For example, the event *applying* results in the action *apply*. The actions are the services the Object class provides. Action names label the heads of arrows, representing transitions, to indicate that it is to be executed when the next state is entered. The transition is made when the named event is generated by an action.

  Actions are represented by sequences of instructions. The sequences are typically short and simple because there are many actions distributed among the Object classes.

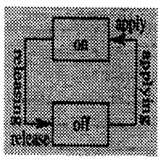
- The *process model* or function model, shown in Example 2.11, represents the system as a process. Each action is represented by an oval labelled with the action name. If the execution of the action results in the generation of an event then a directed arrow is drawn between the oval representing the action and any ovals representing actions which would be executed as a result of a state transition. Data stores used by actions are represented by parallel lines labelled by the variable name.

Example 2.9. Structure models in SD. The structure model for the MUL lift



shows the part of the model corresponding to the brake mechanism as highlighted.

Example 2.10. Behaviour models in SD. The behaviour model and action definitions for the MUL brake

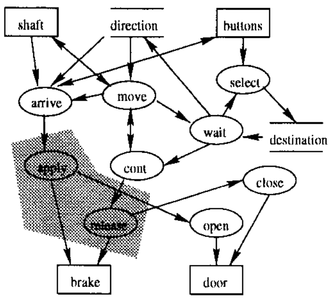


```
apply is
        apply brake;
        generate opening.

release is
        generate closing;
        release brake
```

is typical of the models defining the behaviour of the shaft, door and button classes.

Example 2.11. Process models in SD. The process model for the MUL lift



shows the part of the model corresponding to the brake mechanism as highlighted.

Typically, the construction of the SD models is based on a statement of requirements. In such cases, the software developer uses conventions for transforming the statement of requirements into structure, behaviour and process models. Such a transformation particularly suits requirements stated in formal languages [Bac87, C90, M$^+$88, Dro89] but natural language requirements can also be transformed based on its logical structure [MEGT96]:

- Nouns are transformed into Object classes and attributes.
- Noun phrases are transformed into structural associations between Object classes.
- Verbs are transformed into actions.
- Verbs phrases are transformed into functional associations between Object classes, transitions between states and data-flows between actions.

Such an approach places emphasis on the description of the abstract notions of structure and function, represented within the surface structure of the statement, as opposed to more concrete concepts embodied within the meaning of nouns and verbs [Goo90, Who78].

Once the analysis of the system is complete the design of the software begins. This move from analysis to design is characterized by a shift from the problem domain of the real-world system to the solution domain, consisting of software components with which to build the required system [MP92b]. There are clearly parallels between engineering design in PD [Pug91] and design in SD since both focus on the construction of systems at the component level after the organization of the parts has been established in a previous stage.