Chapter 4

Learning in Computer Science

This chapter explores the potential for EM to be applied to learning specific topics in *databases*, *computer graphics*, and *artificial intelligence*. EM has grown up in a computer science environment. When the idea first arose in the 1980s, even before the term 'Empirical Modelling' was conceived, it was developed by computer scientists and early applications of EM were in software engineering and concurrent systems [Bey07a]. Later on, the first applications of EM to education were for computer science education. Examples include the EDDI database environment that was used for a number of years to teach relational databases at the University of Warwick [BBRW03] [BCY95]. It is natural that the most extensive account of EM as ET in this thesis comes from computer science. The account in the current chapter demonstrates the *experimental*, *flexible* and *meaningful* strands of the *eight significant characteristics of learning* described in Chapter 1. In each of the following three sections a particular strand is emphasised, although there is some cross-over as the eight characteristics are evident in each of these three learning situations. In particular, the section relating to databases explores EM's support for the *meaningful* characteristics of learning, the section connected to graphics examines the *flexible* characteristics, and the section referring to artificial intelligence is concerned with the *experimental* characteristics.

4.1 Meaningful learning in databases

In this section I will show how EM can offer support for teaching a specific topic in relational databases, namely an algorithm for testing lossless join[†]. In particular, I highlight EM's support for the characteristics of the meaningful strand (§1.2.6–§1.2.8) of the eight

[†]Previously discussed in a joint paper presented at the International Conference on Advanced Learning Technologies 2005 in Kaohsiung, Taiwan [BH05b] in which my contribution is the construction of the TLJ model.

Input: A decomposition of a relation R of attributes $A_1, A_2, ..., A_k$ into sub-relations $R_1, R_2, ..., R_n$ and a set of functional dependencies (FD) for R.
Output: True when $R_1, R_2, ..., R_n$ is a lossless decomposition of R.
Algorithm:

1. Create a table with n rows (corresponding to the sub-relations $R_1, R_2, ..., R_n$) and k columns (corresponding to the attributes $A_1, A_2, ..., A_k$).
2. Initialise the table by filling the cell at row i column j with α_j if the attribute A_j is in the relation R_i else β_{ij}.
3. Repeatedly transform the table until no further changes can be made:
   (a) Consider a functional dependency $X \rightarrow Y$
   (b) If two or more rows of the table have matching attributes of X then transform their Y attributes to match:
      i. If α_i is on one row, put α_i in the other rows.
      ii. If β_{ij} is on one row, put β_{ij} in the other rows.
4. If there is a row containing all α's $(\alpha_1, \alpha_2, ..., \alpha_k)$ then the decomposition of R into $R_1, R_2, ..., R_n$ sub-relations has a lossless join.

Figure 4.1: The Testing Lossless Join (TLJ) algorithm.

significant characteristics of learning outlined in Chapter 1.

4.1.1 An algorithm for testing lossless join

In relational database design, as described by Ullman in his book *Principles of Database Systems* [Ull82], it is important to be able to determine whether a decomposition of a relation R into two or more smaller relations $(R1, R2, ...Rk)$ results in a loss of information (which would usually be undesirable). A particular decomposition has a lossless join if, for any given extension r of R satisfying all the functional dependencies (FDs) that hold in R, the natural join of the projections of r onto each of the k sub-schemes is r itself [Ull82:p227]. The Testing Lossless Join (TLJ) algorithm, as specified by Ullman [Ull82:p227], is a standard component of the relational database theory that can be used to find out if a particular decomposition is lossless. The algorithm is described using pseudo-code in Figure 4.1.

For the purposes of this exposition, without loss of generality, all FDs will be assumed to be of the form $X \rightarrow S$, where S is a single attribute. The representation of the entries in the table can also be simplified so that they have numeric values. Specifically, as and bs can be represented by integers: each a by 1 and the initial b entries by integers greater than

1. When several values are to be equated, it is then appropriate to equate all values to the least. This representation, which is suited to computer implementation, is valid since the indices of as and bs are redundant, and all comparisons are made between elements in the same column.

4.1.2 Educational technology for the TLJ algorithm

The TLJ algorithm is in some respects a natural target for computer support. For instance, a student who is exercising the algorithm (or a teacher who is demonstrating the algorithm) typically indicates the successive modifications that are made to the array by crossing out entries and inserting their new values until such time as the array entries become difficult to read, then making a new copy of the array and repeating the annotation process. This is an error-prone process that does not always give a clear indication of the precise steps carried out or the reasoning behind each step. It is easy to see that, when we consider the possible motivations and issues that arise in learning, presenting or assessing the TLJ algorithm, the list of requirements becomes very large. The teacher alone will typically want: a dynamic way of presenting the algorithm that draws attention to the specific observations and actions that are being carried out at each step; to be able to simulate the operation of the algorithm in full; to be able to experiment with different sets of FDs, perhaps even whilst the algorithm is being executed; to emulate errors that a student might make in exercising the algorithm; to use the model as the basis for exercises that test a student's understanding as comprehensively as possible. In devising exercises or an examination question, the teacher will not wish to restart the algorithm from scratch at each new iteration required in the design. The range of situations in which the teacher could exercise the algorithm is vast, and a complete description of all the situations is impossible to specify prior to interaction. Unless the teacher enjoys the dedicated support of a developer, they will ideally want to be able to adapt the model relatively painlessly themselves to take account of different perceptions of what the student requires, and of any special, possibly idiosyncratic, misunderstandings they have.

As a requirement for a conventional program, this presents a formidable challenge. What is more, it is quite apparent that the requirement is not in any sense complete. In constructing a conventional program to meet these needs, there will invariably be optimisation for specific purposes that will prove obstructive to future extensions. The key to addressing this problem is to recognise that what is required of a model to support

the learning of the TLJ algorithm is a form of automation that can integrate fully with the activities that the teacher can perform manually as the need arises. The teacher (as a learner) should be able to attend to personal interests or concerns (§1.2.6), new situations or contexts as they arise (§1.2.7), and previous experience as it continually evolves (§1.2.8). In effect, this allows human discretion and intelligence to be exercised in situations where there is no satisfactory preconceived fully automated solution that can be applied. This is the function of the EM construal for the TLJ algorithm to be described below.

4.1.3 EM support for the TLJ algorithm

The primary observables in the TLJ algorithm are the contents and attributes of the table that is generated in executing the algorithm and the FDs that are associated with these attributes. Both teacher and student come to understand the algorithm in terms of just these observables; building a construal to embody these observables, and the patterns of dependency and agency to which they are subject, is also a most appropriate way for the developer to provide support for the manual, semi-automated or fully automated interaction that must accompany the learning of the algorithm.

Learning the TLJ algorithm is linked to a pattern of observation that applies at each iteration. The learner consults the current state of the table with a specific FD $X \rightarrow S$ in mind, observes the pattern of tuples that arises in the columns associated with the left-hand side X of the FD to detect where there are duplicates, then observes how this pattern applies to the column associated with the right-hand side S of the FD. The core step of the algorithm is the substitution of the resulting transformation of the column associated with S for the original column.

For a particular table and FD, the above ingredients of the core pattern of observation can be displayed pictorially as in Figure 4.2. The arrows in this figure represent dependencies between observables, expressing the way that a given state of the TLJ table, and a given FD determines the set of columns LHS and a column RHS, and how the duplicate rows in the set of columns LHS then determine the updated entries in the column RHS. In the modelling environment used to develop the construal, these dependencies can be directly specified and are automatically maintained (as discussed in §2.2.2). This makes it possible to explore, in a meaningful fashion by tracing the dependencies, the way in which the current instance of this pattern of observation is affected by changing the current state of the TLJ table, or the current FD.

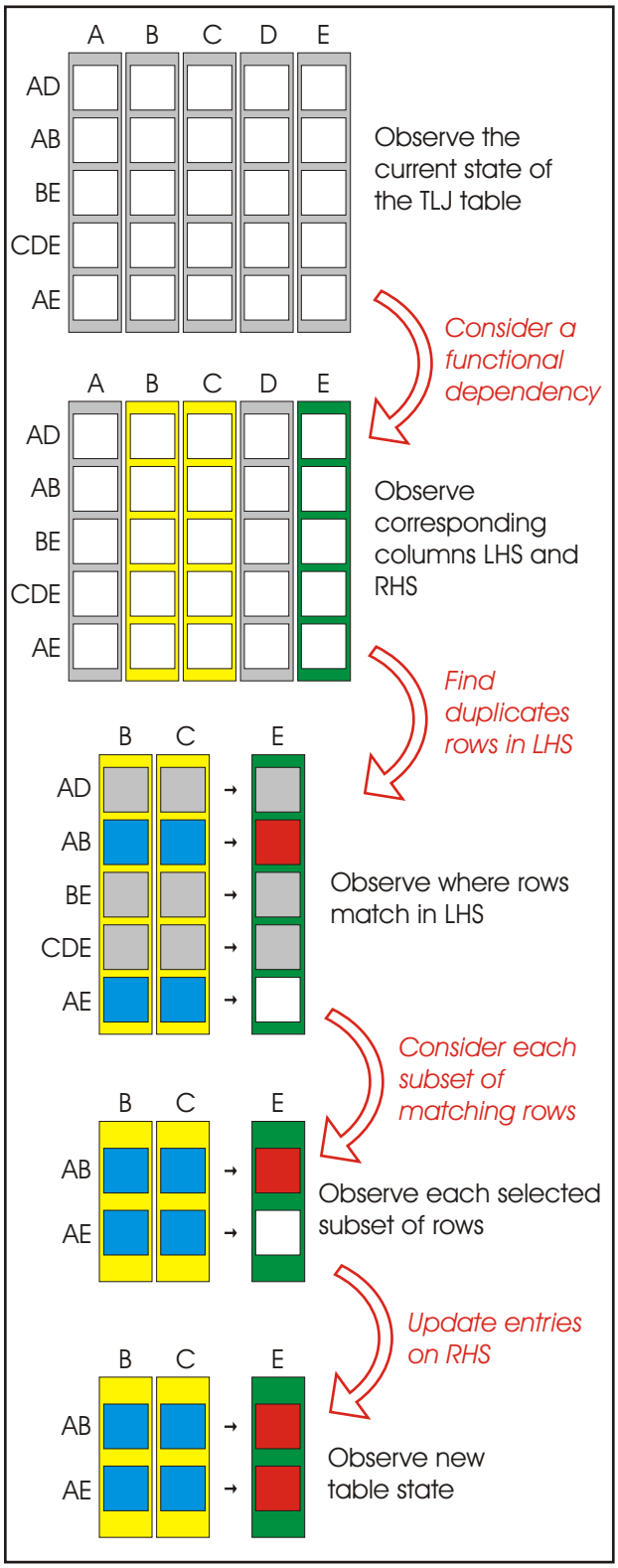


Figure 4.2: Steps in the TLJ algorithm.

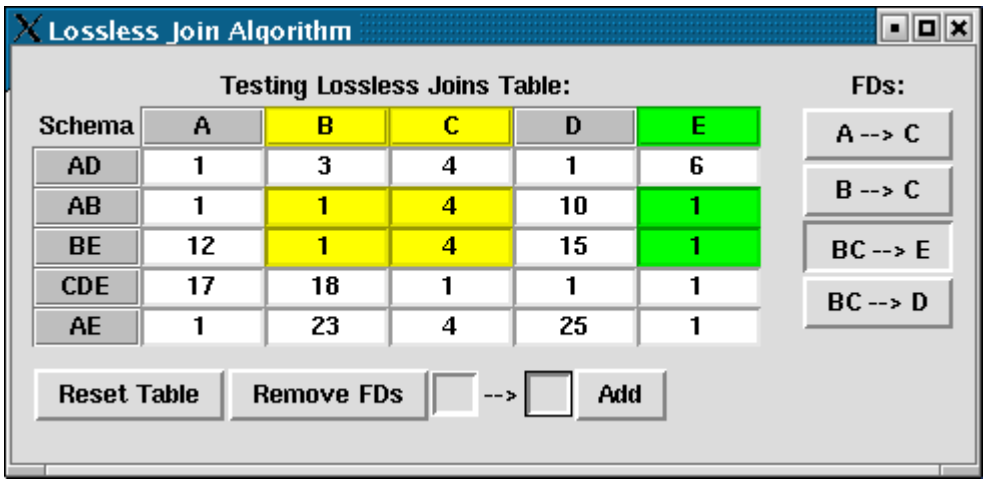


Figure 4.3: The TLJ construal.

4.1.4 Developing and deploying the construal

The exploratory activity that surrounds the identification of observables and dependencies is a core activity that is central to the interests of the student, the teacher and the developer. As Figure 4.2 illustrates, the contexts for observation with which the student must become familiar in learning the TLJ algorithm are rich and subtle: they involve moving from global observation of the entire table to localised observation of the entries in specific rows and columns. It is also significant that the activities denoted by the arrows in Figure 4.2 are best conceived as mental operations on the part of the student, preparatory to the action of updating the table. From a teacher's perspective, each of the arrows can be interpreted as a link in a chain of observation involved in executing a step of the TLJ algorithm. As such, it can be the subject of an exercise: for instance, identifying the columns LHS and RHS, given a table and a FD. Decomposing the pattern of observation into a chain of simpler observations also has potential value as a diagnostic tool: for instance, helping the teacher to detect where a student understands the updating mechanism correctly, but is mistaken in their interpretation of a FD relation. The TLJ construal can be exercised in ways that are personal to the teacher or the student reflecting their specific needs or interests—supporting the characteristic of *learning as motivated by personal interest* (§1.2.6)—and in new situations or contexts as they arise—supporting the characteristic of *learning as a situated experience* (§1.2.7).

There is a very direct correspondence between Figure 4.2 and the EM construal for the TLJ as that depicted in Figure 4.3. This correspondence is best appreciated by interacting with the `tkeden` environment, but it is to some extent apparent from the relationship between Figure 4.2 and Figure 4.4. Just as the pattern of observation depicted

```
project_table_LHS_FD is project(current_table,
    makestrlist(FDs[current_FD][1]));

project_table_RHS_FD is project(current_table,
    [FDs[current_FD][2]]);

pattern_duplicate_rows is index_duplicated(
    tail(project_table_LHS_FD));

newcol is transformcol(makelistcol(project_table_RHS_FD),
    pattern_duplicate_rows);

newtable is apply_current_FD_current_table(current_table,
    newcol);
```

Figure 4.4: Observables and dependencies in the TLJ construal.

in Figure 4.2 is the core of the TLJ algorithm, so the script of five definitions linking observables and dependencies in Figure 4.4 is the core of the TLJ construal. The names of the observables in Figure 4.4 have been made more expressive, and the code for operators (such as index_duplicated, and makelistcol) has been omitted, but the definitions are essentially as they appear in the `tkeden` script. The correspondence between Figure 4.2 and Figure 4.4 demonstrates the extent to which an experience of the algorithm can be loosely connected to an experience of observing the dependencies in the TLJ construal. EM's support for *learning as a continuous experience* (§1.2.8) can be recognised from the necessary close correspondence between observations in the world and observations in the construal. In order to further illustrate the use of EM for liberating the *meaningful* characteristics of learning, a brief explanation of how this script was developed, and relates to the pattern of observation in Figure 4.2, is appropriate.

As is evident by inspection, the values of all the observables in the script in Figure 4.4 are determined from the index of the FD that is currently of interest (`current_FD`) and the current contents of the TLJ table (`current_table`). The first two definitions determine the contents of the columns that correspond to the LHS and RHS of the current FD respectively. The third definition identifies the pattern of duplicate rows in the columns in the LHS of the FD; the fourth expresses the way in which the new contents of the RHS column is to be updated by consulting the pattern of duplicate rows. The final definition expresses the relationship between the original value of the table and the value that it takes after the FD has been processed. These definitions correspond closely to the links in the pattern of observation in Figure 4.2: in establishing the definitions using the `tkeden` interpreter, the operators introduced to specify the relationship associated with each link are tested in isolation by supplying different test values for the parameters in much the

same way that the student might confirm that they have understood each observational link in mastering the algorithm. Though the development of a script may seem to have more of the characteristic flavour of conventional programming, it remains anchored to the learning domain. The missing elements of the `tkeden` script are the specifications of the operators themselves, which take the form of rather straightforward procedural code to compute an output from an input without side-effect. The script illustrates other features that are of interest from a computational perspective. These include:

- the re-use and adaptation of standard operators (such as the relational operator project, borrowed from the relational database extension of `tkeden`).
- the use of definitions to maintain dependencies between different modes of observation that are a common concern for traditional programmers, namely those that are associated with two or more data structures for a particular application (such as the conversion function `makelistcol`).

For the experienced developer using `tkeden`, the model-building task is greatly simplified by a combination of these three techniques: creation of relatively simple functions without side-effects; re-use of existing functions and scripts; and the use of definitions to maintain many different consistent concurrent representations of a given family of observables. The interface shown in Figure 4.3 was constructed using observables and dependencies in much the same way as the explicit TLJ-related observables in Figure 4.4. The GEL (Graphical Environment Language) notation [EMP:gelHarfield2007], developed by the author, was used to construct the buttons and labels which for the purpose of this model were dependent on the `FDs` and the `current_table` observables. The buttons semi-automate the actions that a student may be interested in performing, such as changing the current FD or changing the set of FDs (corresponding to a modification of the `current_FD` or `FDs` observables that are used in Figure 4.4). Figure 4.5 introduces the basic principles of GEL, and further information can be found in the GEL documentation [EMP:gelHarfield2007].

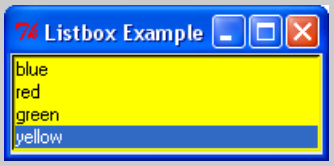
4.1.5 Implications for learning

In the above discussion of the TLJ construal, it has been shown that EM offers a suitable approach to technology-enhanced learning according to the three characteristics of the *meaningful* strand of learning as set out in Chapter 1, that is:

The Graphical Environment Language: GEL

One of major technical contributions to this thesis is the creation of the GEL notation [EMP:gelHarfield2007]. GEL is a notation that has been added to the `tkeden` environment enabling the creation of new graphical environments or graphical user interfaces. GEL itself is a model that was created using the agent-oriented parser (AOP) inside `tkeden`. The AOP is another of my contributions to the EM project [EMP:agentparserHarfield2003] that can be used to create new notations. The GEL notation is open to the same exploration and experimentation as other models, including models created using GEL. Much of the development of the GEL notation occurred on-the-fly whilst I was using the GEL notation.

GEL can be used to create graphical environments that contain general GUI components such as windows, labels, buttons, list boxes, option buttons, radio buttons, scroll bars and scale bars, and `tkeden` specific components such as SCOUT or DoNaLD windows. GEL is a dependency-based notation, enabling complex relationships to be modelled and maintained among elements of the model. The example of a GEL script below demonstrates some simple components and the use of dependency.




```
myexample = window {
      title = "Listbox Example";
      content = [mylistbox] };

mylistbox = listbox {
      selectmode = "browse";
      items = [ "blue", "red", "green", "yellow" ];
      selecteditems = [ "red" ];
      height = mylistbox_items#;
      background = mylistbox_selecteditems[1] };
```

The first definition describes `myexample` as a window with a title and containing something called `mylistbox`. Entering this definition into the `tkeden` interpreter will create a new window on screen. By dependency, when `mylistbox` changes, the window will be updated. The second definition describes a listbox with initially four items. Entering this definition will create the listbox inside the window. The height of `mylistbox` is dependent on the current number of items, therefore if another item is added by redefining `mylistbox_items` then the listbox height will change and also the window will be updated. The final line in the definition of the listbox describes the background as dependent on the first selected item in the listbox. Thus, if the modeller selects the 'yellow' item then the background of the listbox becomes yellow, as shown above.

The GEL notation has been used to construct a number of models in this thesis. Further information on GEL can be found in the documentation and in an interactive introductory model [EMP:gelHarfield2007].

Figure 4.5: Description of GEL.

- learning is motivated by personal interest (§1.2.6);
- learning is a situated experience (§1.2.7);
- learning is a continuous experience (§1.2.8).

The teaching of a specific topic in databases such as the TLJ algorithm requires support that is sensitive to the personal needs or interests of the teacher or the student, and to the variety of situations in which the algorithm can be used for learning. In some cases the teacher may be interested in exercising the algorithm very general ways as a device in a lecturer, and in others the teacher may exercise a very specific element of the model for the benefit of a particular student to rectify a misunderstanding. Furthermore, the TLJ construal can be seen as a device for navigating experience according to the EFL as described in §2.2.8. The difficulties which learners encounter with the TLJ algorithm involve a mismatch between formal knowledge of the algorithm (the bottom of the EFL) and practical experience of the algorithm (the top of the EFL). EM supports *learning as a continuous experience* (from the top of the EFL to the bottom) because artefacts such as the TLJ construal enable learners to exercise the algorithm on a practical level—personally making the interactions involved—and on a formal level—observing the dependencies leading to state-change.

The difficulty of unifying the roles of *student*, *teacher* and *developer*, as discussed in §3.3, is one of the obstacles to a constructionist approach to technology-enhanced learning. The technical problems of supporting the degree of openness in interaction that constructionism ideally presumes are so acute that Ehrmann [Ehr00] has been led to question whether the vision of learners constructing their own learning environments is a mirage. It is clear that in activities such as developing micro-worlds for children–at any rate with current software tools—there is little prospect that the learners can themselves carry out the model construction. The TLJ case study is of interest because it proves that in principle there can be a high degree of synergy between interactions that are demanded of the learner in the roles of student, teacher and developer. For the target group of learners (viz. undergraduates with high levels of programming skill following an advanced module in database theory), there is no great conceptual or practically significant distinction between the kind of activity involved in learning about the lossless join algorithm and that involved in constructing the associated EM construal. It remains to be seen to what extent, subject to appropriate tool refinement and suitable training in the application of

EM principles and tools, the same synergy between learning and model-building can be demonstrated in other learning contexts.

4.2 Flexible learning in computer graphics

In this section I will show how EM tools and principles can offer support for the *flexible* characteristics of learning discussed in Chapter 1, in the context of teaching a specific topic in computer graphics. The characteristics of 'learning occurring without a prescribed outcome' and 'learning not following a preconceived path' are evident in EM because the learner is a model-builder who can: reuse models in different contexts; combine models together; take on the unified role of student, teacher and developer. The models explored in this chapter have led to one of my main technical contributions which is the construction of a generic presentation environment for EM.

4.2.1 Flexibility through model reuse

Code reuse is important in programming when you have two or more tasks that are similar or the same and therefore a part of the code can be shared or reused. In model-building, reuse can have a slightly different meaning. Model reuse involves taking an existing model and fitting it to a new context (as introduced in §2.2.7). The fitting often entails the extension or reduction of the model. Similarities can be drawn with Piaget's theory of learning [Pia50], discussed in §1.2.2, as necessarily involving assimilation or accommodation, where a new experience is fitted to existing knowledge, or knowledge is modified to fit a new experience. In the case of model-building, either the model can be modified to fit a new context or the model-builder's imagination may have to be stretched or modified to fit the limitations of an existing model. A good example of reuse from the EM archive is the jugs model [EMP:jugsBeynon1988] which has been reused in a wide variety of contexts. Initially developed as an artefact for explaining the concept of dependency, it has subsequently been used, as discussed in §3.2, for modelling queues at a bar, learning a language, and displaying chords on a violin [EMP:kaleidoscopeBeynon2005], as well as the basis for a small concretisation case study[†].

[†]This was part of a joint paper entitled *Varieties of concretisation: an illustrative case-study* that I presented at the Baltic Sea Conference on Computer Science Education (Koli Calling 2005) [BHJ05], but it is not discussed further in this thesis.

4.2.1.1 The 3D room viewer

A group of models relating to a simple room demonstrate an example of model reuse over a long period of time. Each of the models is based on the idea of modelling a room containing a table, a lamp on the table, and various other objects. The model demonstrates dependencies such as the position of the lamp being related the position of the table. The original room was the first model that used the line drawing notation DoNaLD by Edward Yung [Yun90] [EMP:roomYung1989] in 1989 and remains a popular model for introducing dependency. It was further refined in 1991 by Simon Yung and integrated with the SCOUT notation which added viewport capabilities that enabled the room to be embedded in a viewer model [Yun93] [EMP:roomviewerYung1991]. Both models are 2D top-down views of a room. The focus of this chapter is the 3D room viewer which enables the room to be viewed from any angle as a 3D line drawing with perspective (see Figure 4.6). The 3D room viewer was a 3rd year project by Andy MacDonald in 1997/8 [Mac98] [EMP:room3dMacDonald1998]. Beyond the room viewing aspect, the model gives the user the ability to apply forces to the objects in the room using the circular force control in Figure 4.6. The 3D room viewer also drawns on an earlier model by Richard Cartwright for creating 3D line drawings in DoNaLD [BCC96]. When the SASAMI notation was developed for creating 3D artefacts in OpenGL, another 3D room model was constructed by Carter [Car99] [EMP:room3dsasamiCarter1999]. Figure 4.10 on page 103 illustrates the history of the room model to be explored further in this section.

In 2007, the 3D room viewer [EMP:room3dMacDonald1998] was used by Meurig Beynon for demonstration purposes in a computer graphics module for undergraduate computer scientists [Bey07b]. Apart from the reuse of the previous room viewer models, this shows a concrete example of a reused model that can perform a function in a completely different context to that initially envisaged by MacDonald in his 3rd year project [Mac98]. MacDonald was interested in applying forces to furniture in the room, whereas for the purposes of Beynon's lectures such facility was less relevant than the part of the model concerned with presenting a 3D scene. In the lectures, the model was used to explore issues surrounding the transformation of a 3D scene into a 2D view through experiments with definitions connected to the projection function. Such reuse is behind the characteristic of EM activity as not being concerned with prescribed outcomes for models—they can be used for a wide range of activities leading to outcomes that were not prescribed beforehand. Apart from the model's function as a demonstration tool, it

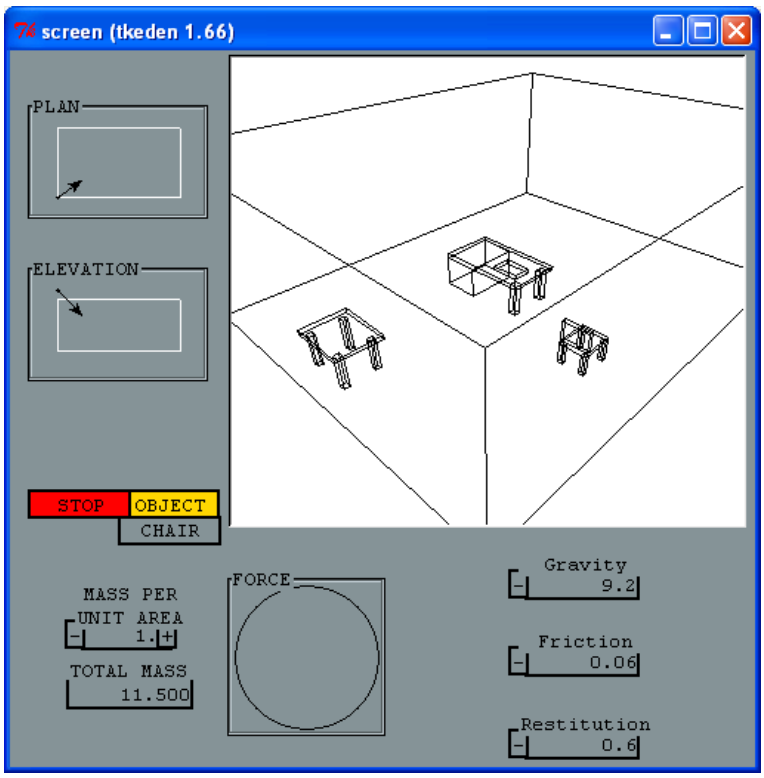


Figure 4.6: The original 3D room viewer model.

could also be downloaded by students to explore and experiment by themselves. Beynon's interactions in the form of a walk-through provide a basis from which to flexibly explore the transformation from a 3D model to a 2D drawing [Bey07b]. Furthermore, the model is open-ended in that once a student becomes more familiar with it, the model could be reused by a student in a different context in order to make sense of another topic. For example, the 3D room model could be used to experiment with different clipping algorithms for drawing lines [HB86]. In this way the student is a model-builder exploring how principles from books/lecture notes on computer graphics relate practice in the world.

Beynon's detailed walk-through [Bey07b] of the 3D room model is a special type of reuse because no new model was created—the walk-through purely uses MacDonald's model [EMP:room3dMacDonald1998]. Many of the examples of model reuse in EM have involved fairly trivial models (e.g. [EMP:jugsBeynon1988], [EMP:roomYung1989]) and possible reasons for this may be that complicated models require too much effort on the part of the new model-builder in order to understand the model. Beynon's walk-through is unique in that it provides material, in the form of example interactions, to prompt the discovery of interesting aspects of the model, specifically in this case for understanding a topic in computer graphics. In some respects this has enabled the type of learning that *need not have a prescribed outcome* and *does not follow a preconceived path.*

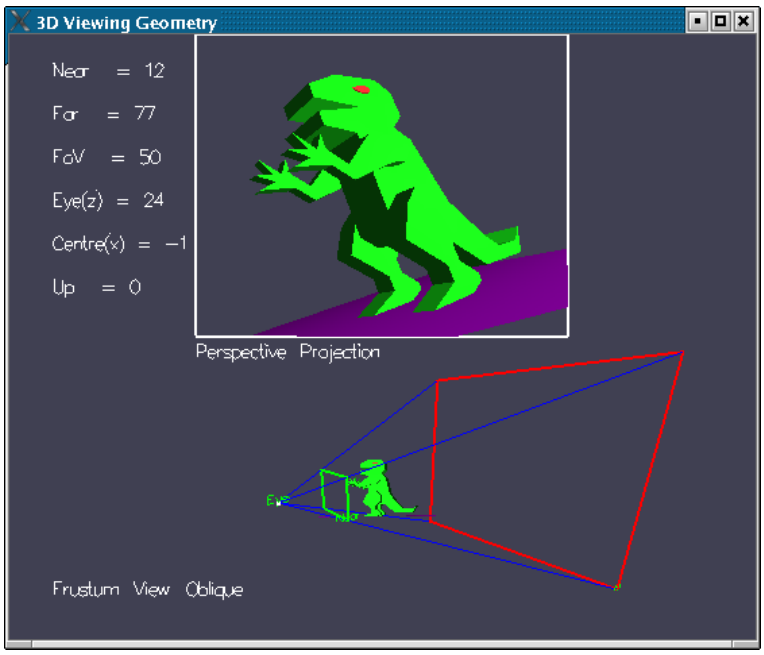


Figure 4.7: A C/C++/OpenGL program for exploring 3D to 2D transformation.

4.2.1.2 Contrast with the 3D Dino Viewer tool

Previously, another tool was used for demonstrating 3D to 2D viewing transformations in the computer graphics module. The 3D Dino Viewer tool, developed by Abhir Bhalerao [Bha06] using C/C++/OpenGL, was designed to demonstrate the transformation from a 3D scene (of a simple dinosaur) to a 2D view. The tool allows the learner to explore the effects of changing the eye position, the eye direction and the front and back clipping planes (see Figure 4.7). The results of these changes are displayed in two 3D views: from the perspective of the eye, and from the perspective of an external observer. The tool makes use of dependency to maintain a consistent state for example between the values of the viewing plane vector and the 3D view of the scene. In some ways, this is an example of Empirical Modelling principles as there is a direct dependency between the values of the transformation parameters and the two OpenGL views. The tool has some of the qualities of a spreadsheet discussed in §3.4 where changing the values in a cell causes related cells and graphs to be updated simultaneously and automatically.

The 3D Dino Viewer is well aligned to the material in the module and the specific topic of 3D to 2D transformation in computer graphics [HB86]. In terms of features and usability, the artefact is an excellent example of educational technology for communication purposes. It can be used by a lecturer for demonstration and by students for post-lecture experimentation. It is particularly easy-to-use in terms of exercising functionality, much easier at least than the 3D room viewer models described above. Part of the reason for

this is in the way that it was implemented. Bhalerao developed the 3D Dino Viewer with one particular task in mind: demonstrating 3D to 2D transformations. As with most programming tasks, the closed-world approach to the implementation allows the programmer to tune the application for efficiency and ease of use. This results in an easy-to-use artefact with a very specific use. Herein lies a particular issue that stresses the paradigm conflict between learning in the world and learning through computers: the artefact is so easy to *use* that there is no motivation for learners to *experiment* with it in ways that may lead to learning spanning the depth of the EFL (e.g. a continuous experience from practical understanding to formal knowledge). Moreover, because the artefact has been made so easy-to-use with specific functionality, it has certain prescribed behaviours that would be *too difficult* for most learners to flexibly *extend* or *reuse* without an extensive knowledge of C/C++. This means that there is little potential for the learner to explore the topic of computer graphics further using the 3D Dino Viewer as it forces a predefined path of interaction and a specific outcome for learning. Learning with the 3D Dino Viewer follows a preconceived path and has a prescribed outcome, and therefore it does not qualify for satisfying the *flexible* strand of characteristics set out in Chapter 1.

The idea that a computer-based artefact might be *too easy-to-use* seems contrary to the common 'push' to make computers more accessible. I am not advocating that computers should be more difficult to use, but that we should not sacrifice imagination and flexibility for the sake of usability. In many cases computers can be made more user friendly, and this should be encouraged, but not to the detriment of what can be achieved with computers. In terms of learning, the idea of making things 'easier to use' in computing often gives rise to the idea that educational technology can make subjects and skills 'easier to learn'. As Jonassen [Jon06] argues, good communication of information does not necessarily lead to learning, and 'easy-to-use' artefacts do not readily imply 'easy to learn' subjects and skills. Can you imagine an easy-to-use piano that you could learn to play adeptly after using on your first time? (This seems to be what some educational technology aspires to do!) The truth is that learning to play any musical instrument is difficult and requires skill, effort, perseverance, and practice. The motivation and satisfaction comes from progression, it comes from getting it wrong many many times, being flexible in ways of use, and discovering how to get it right. When learning a musical instrument, there is this depth to the learning. If something is too easy then it does not offer the same learning experience as an experience that has involved a deeper understanding.

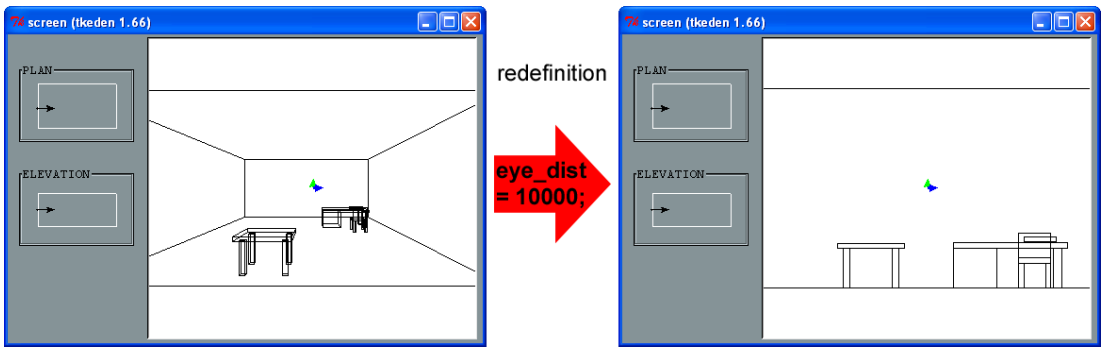


Figure 4.8: Redefinition to change the projection from perspective to orthogonal.

If we 'let go' of the technologist's agenda of making computer-based artefacts easy-to-use then learners might progress to a deeper learning. For educational technology that supports *experimental*, *flexible* and *meaningful* learning technologists should be striving for environments where models can be constructed and manipulated. Such sentiments are expressed by Jonassen that if we cannot construct a model of something then we do not understand it [Jon06]. In the general terms of modelling, as Morgan & Morrison point out in *Models as mediators* [MM99], model use is much less helpful than model construction and experimentation.

Although the EM 3D room viewer may be more difficult to use, if the learner has the motivation to attempt to make sense of the model then it provides a much richer environment for learning about viewing transformations. The evidence for this is in Beynon's walk-through of the model [Bey07b], that shows how the model can be exercised in many different ways to highlight issues relevant to geometric constructions. In one part of the walk-through, the 'project' function is dissected to try to better understand what it does. Small changes to different definitions related to the 'project' function are suggested to the model-builder in order to observe the effect. For example, by changing the eye distance in the project function to a negative number, the camera view can be transformed so that it is inverted [Bey07b]. A more useful redefinition, as depicted in Figure 4.8, would be to make the eye distance very large which imitates orthogonal projection, as opposed to the original perspective projection [Bey07b]. Figure 4.8 demonstrates that such changes are not necessarily easy, a redefinition typically requires: a knowledge of the observables in the model (e.g. `eye_dist`), an appreciation of the current state of the model (e.g. `eye_dist = 100`), an understanding of the perceived dependencies in the model (e.g. `eye_dist` is related to the 3D drawing of the room), and an ability to modify state in the required notation.

```
_x_pos = _y_pos = 0;
for (i=1; i<=100; i++) {
    _x_pos = _x_pos + 3.7;
    _y_pos = _y_pos + 1.5;
    eager();
}
```
Panning

```
_x_dir is 500-view_cen[1];
_y_dir is 400-view_cen[2];
_z_dir is 0-view_cen[3];
_x_pos = _y_pos = 0;
for (i=1; i<=100; i++) {
    _x_pos = _x_pos + 3.7;
    _y_pos = _y_pos + 1.5;
    eager();
}
```
Tracking

```
_xpos=370; _y_pos=150;
_z_pos = 100; _z_dir = 0;
_y_dir is sin(swivel_angle);
_x_dir is cos(swivel_angle);
for (i=1; i<=100; i++) {
    swivel_angle = i*2*PI/100;
    eager();
}
```
Scanning

Figure 4.9: Three possible interactions with the 3D room viewer.

The 3D Dino Viewer tool designed by Bhalerao already includes a feature to switch between perspective and orthogonal projection. It is a simple button press, but it does not show the relationship between perspective and orthogonal in terms of eye distance as the 3D room viewer model does—that orthogonal is like perspective at a large eye distance. This was not a feature that was consciously built into any of the room viewer models, but a consequence of the model being constructed with attention on observables, dependency and agency (as described in §2.2.2). The distinction between conventional software development and EM (as demonstrated in Chapter 3) is evident in comparing the 3D Dino Viewer and 3D Room Viewer—the former is a black-box with a closed set of prescribed behaviours and the latter is an artefact that can be taken apart to explore the relationships that lead to observed behaviours. As a result, the EM model can be explored in all manner of ways that have not been preconceived. One such example is animation which, as Beynon suggests in the walk-through, can be achieved by redefining the view plane parameters as in Figure 4.9. The model-builder can simulate actions that can be performed with a camera, including: panning around a scene; moving the camera whilst directing it at a fixed position; and scanning in every direction from a fixed camera location. The scripts, taken from Beynon's walk-through [Bey07b], for each of these interactions are shown in Figure 4.9.

In summary, the 3D Dino Viewer is a tool with a particular job in mind: communicating the effects of pre-specified parameters for transforming a 3D scene into a 2D view. It is

easy-to-use, but not at all flexible beyond its preconceived behaviour. The 3D room viewer model is an artefact that is open to any manipulation limited only by the imagination of the learner or model-builder. Although it might require more effort on the part of the learner, it can be used to exercise preconceived uses and explore new contexts that fit the needs of the learner. The 3D room viewer model is both a teaching instrument that can be used to give demonstrations in the lecture, and also as a learning environment that the student or teacher can experiment with in his or her own time.

4.2.2 Flexibility through model combination

Another relevant feature of EM, closely related to model reuse, is model combination—reusing two or more models together in combination. In this section I am going to show how this technique can be used effectively to extend the 3D room viewer model and make it more accessible to students and teachers. I will show how several models that I have developed can be combined to create a presentation environment, and later I will combine this with the 3D room viewer. This section briefly explores some exercises in combining and extending models that have resulted in extensions to the `tkeden` tool (i.e. the Agent-Oriented Parser [Har03], the Graphical Environment Language [EMP:gelHarfield2007], the `%html` notation and environment). This eventually leads to the combination of all these models which results in the EM presentation model. Later I shall explore how the presentation model was combined with the 3D room viewer by MacDonald [EMP:room3dMacDonald1998] and the 3D room viewer walk-through by Beynon [Bey07b]. Figure 4.10 illustrates the stages of model-building activity relating to the room viewer models.

4.2.2.1 The `%html` notation and environment

The GEL notation (see Figure 4.5 on page 93) is an example of a notation that I developed as part of this thesis, using the agent-oriented parser (AOP) [EMP:agentparserHarfield2003]. Another notation I have developed using the AOP is the `%html` notation, which forms the basis of the HTML Environment [EMP:htmlenvHarfield2007] that is constructed with a combination of both the AOP and GEL. The motivation for developing the `%html` notation and environment was itself as a tool for model combination. A problematic but essential characteristic of EM tools is that they are used to construct models which are personal and unique to the model-builder (as discussed in §2.2.6). This makes it difficult for other

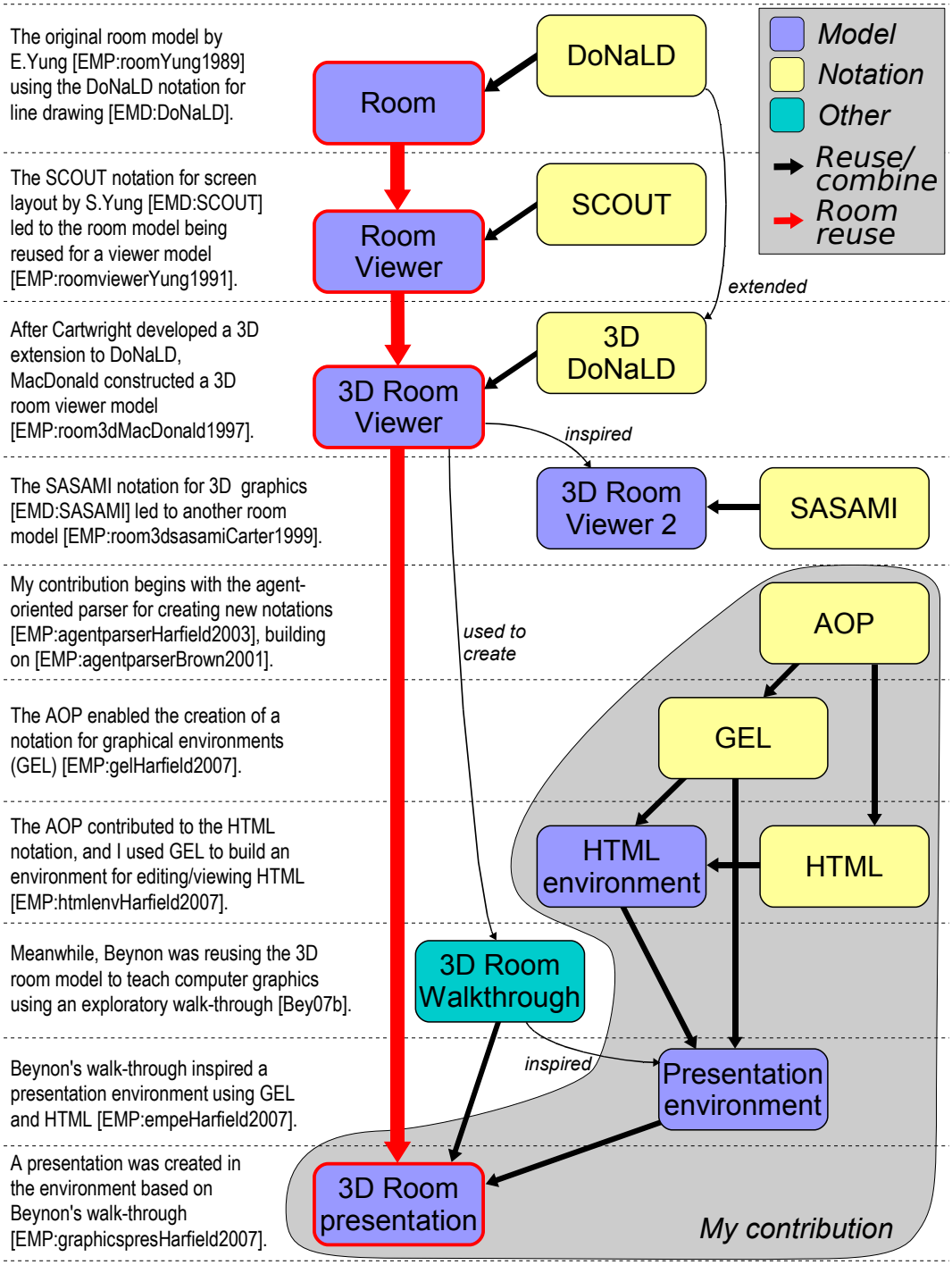


Figure 4.10: The ancestry of the 3D room graphics presention.

model-builders to exercise, explore and extend these models [Kin07]. To improve this situation, it is useful for a model-builder to document and explain the model, for instance by highlighting key observables and indicating interesting redefinitions as can be found in Beynon's walk-through of the 3D room viewer [Bey07b]. The documentation could be part of the model itself, even dependent on the state of the model [Kin07]. The `%html` notation is a solution for creating HTML-based documentation that is linked to the model, and that can be created as part of the modelling activity.

The notation, written in the AOP, parses an extended form of HTML which contains the usual standard HTML tags, and (in the initial version) two further tags: `<eden>` and `<page>`. Generally, HTML transferred across the Internet is interpreted once before it reaches the browser—as depicted in Figure 4.11(a)—and subsequent processing requires the user to request the page again. HTML is handled in a distinctly different way in the HTML Environment [EMP:htmlenvHarfield2007] through the use of dependency. The two tags enable dependencies within an HTML page that leads to the page continually responding to changes in the underlying observables as shown in Figure 4.11(b). Unlike client-side scripting (e.g. Javascript) and server-side programming (e.g. Perl, PHP, ASP), pages reflect the current state of the variables in the environment as opposed to the state at the time the page was sent to the browser[†]. The `<eden>` tag is for making the HTML page dependent on observables/expressions within eden. For example:

```
%eden
contentA = 2;
%html
<p>The content of jug A is <eden>contentA</eden>.</p>
```

The current value of the HTML observable will be the string `"<p>The content of jug A is 2.</p>"`. A redefinition of `contentA`, such as `contentA = 3;`, will result in a corresponding change to the HTML string: `"<p>The content of jug A is 3.</p>"`. Although the above syntax (for the `<eden>` tag) may look similar to server-side programming, it is very different in its semantics. When including a variable in PHP (see Figure 4.11(a)) using `<?php echo contentA;>`, it is an instruction for the translator (when requested) to write out the value of contentA at that particular point in the HTML code. Figure 4.11(a) shows how a script is interpreted once before it is rendered on-screen. With the EDEN script, the HTML is always consistent with the `contentA` observable; the

[†]Network issues are irrelevant to the discussion as the server could be on the client, or equally the HTML Environment could be run over a network using `dtkeden`as discussed in Chapter 5.

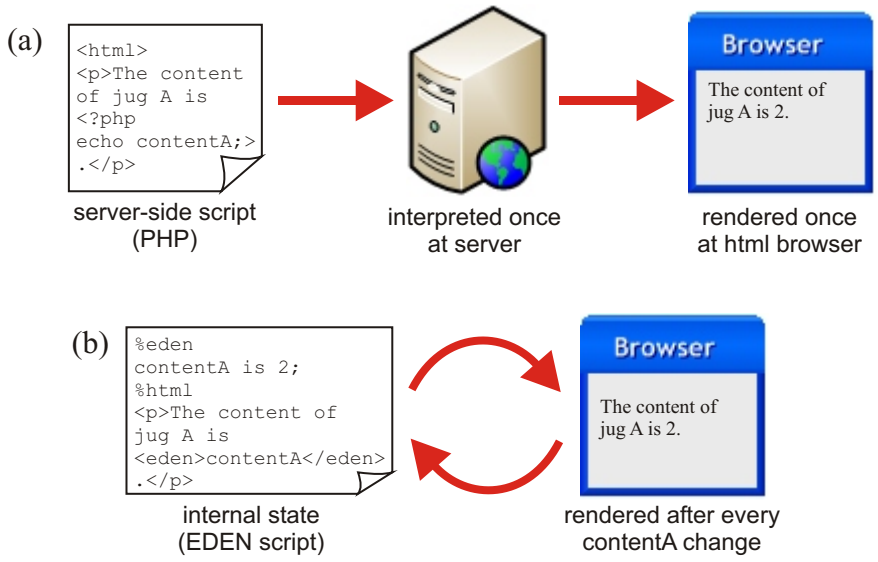


Figure 4.11: Comparing (a) traditional HTML scripting across the Internet with (b) HTML in `tkeden` using the HTML Environment.

HTML will [indivisibly] always shows the current value of `contentA` (see Figure 4.11(b)). An update to `contentA` will cause the HTML script to be updated, hence we say that the HTML script is dependent on `contentA`.

In GEL, a new component was added to parse standard HTML and render it within the component. As explained in Figure 4.5 on page 93 (and in the GEL documentation [EMD]), GEL components have some standard properties (i.e. border, background, relief, etc) and some special properties unique to the component. A HTML component has a property for setting the HTML script, and any changes to this observable cause the component to be updated with the up-to-date rendering of the HTML text. Therefore, text, formatting and graphics within a HTML component are dependent on its HTML script, and the HTML script may be further dependent on other observables within `tkeden`. Thus, if the HTML script is defined as above (`"<p>The content of jug A is <eden>contentA</eden>.</p>"`) then the HTML component will show `"The content of jug A is 2."` as paragraph text. A change to the HTML script will cause the component to re-render the HTML, and likewise, as the HTML script is dependent on `contentA`, a change or redefinition to `contentA` will cause the HTML script to update and hence update the HTML component displayed on screen.

A combined model was created, utilising the HTML notation and the HTML component in GEL, which I have called the HTML Environment [EMP:htmlenvHarfield2007]. This model enables a model-builder to create and view HTML pages that can be used during their modelling activity. Although this model is not very useful as a standalone

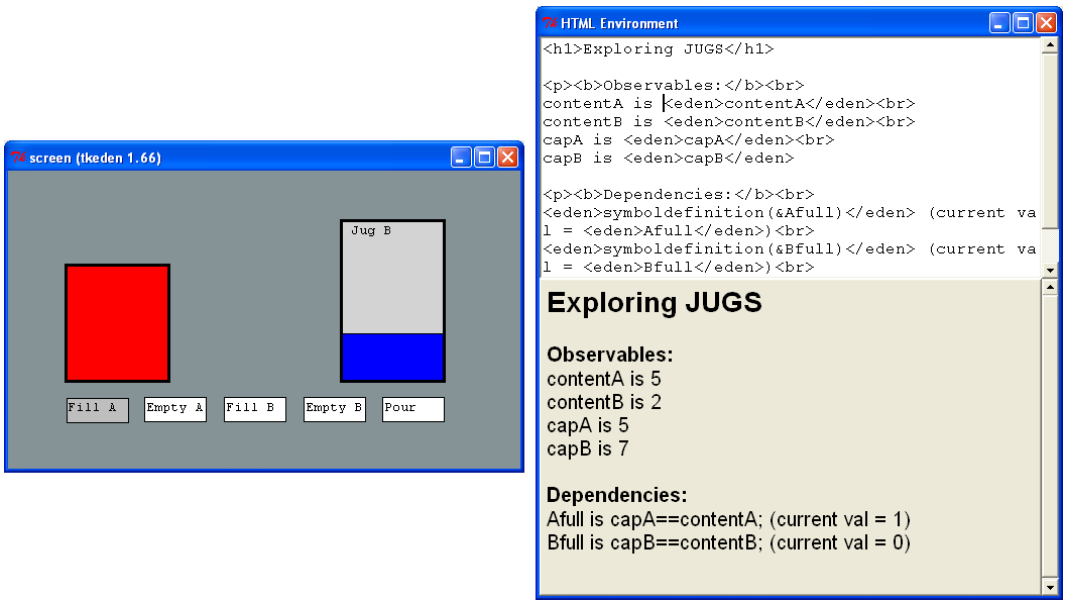


Figure 4.12: The jugs model augmented with an HTML window displaying the current state.

model, it is useful for model-builders to combine with their own models. It's main motivation is to provide support for drawing attention to aspects of a model so that the model is more accessible to other model-builders. Figure 4.12 shows an example of the jugs model [EMP:jugsBeynon1988] with an HTML script displaying the current state of the observables and dependencies. The HTML component, in the lower half of the HTML Environment window, is dependent on the content of the jugs in the left-hand window. King's *Introduction to dependency* presentation [EMP:jugspresentationKing2005] has similar functionality specifically for displaying the state of the jugs model [Kin07]. The benefit of the HTML Environment is that the presentation can be changed on-the-fly by redefining the HTML script.

4.2.2.2 The Presentation Environment

The Presentation Environment (EMPE) [EMP:empeHarfield2007] is an example of using the HTML Environment in a different context. The HTML Environment was created for building HTML pages to document models. The environment was modified to be a presentation tool combining slides with models. The HTML pages became slides that fill most of the window, and other components were added for displaying and interacting with models within the presentation window. Figure 4.13 shows the presentation window containing: a) slides; b) an input box; and c) a SCOUT/DoNaLD interaction box.

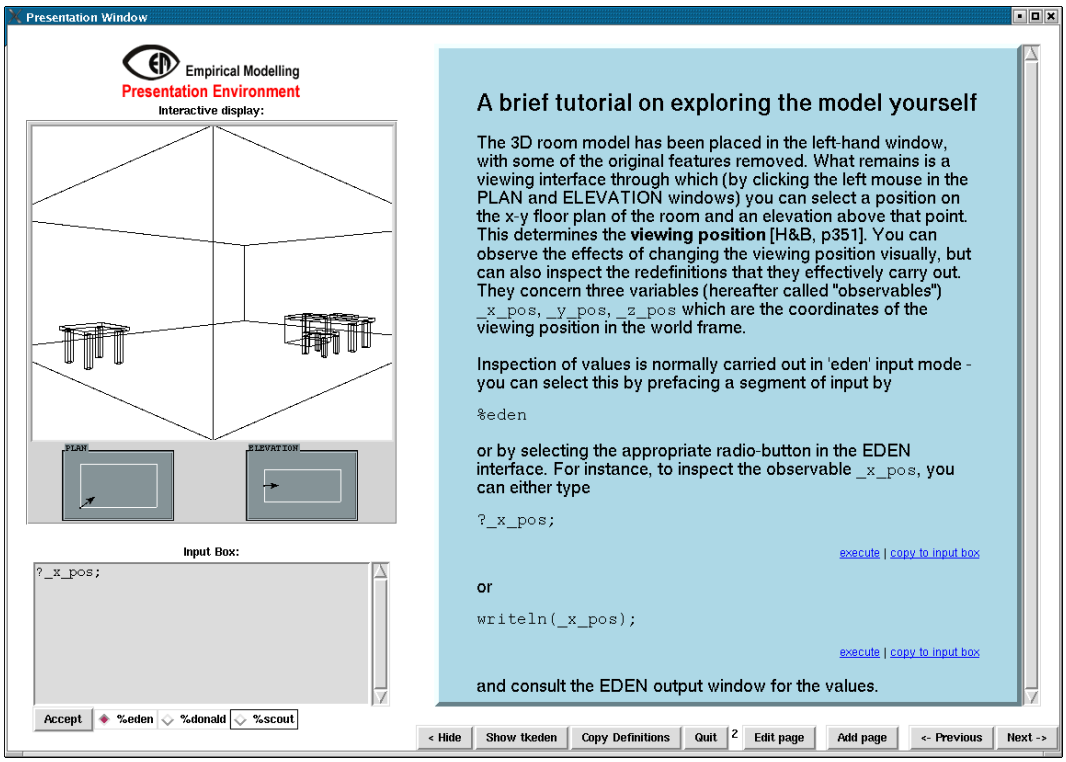


Figure 4.13: The EM Presentation Environment (EMPE).

4.2.2.3 The 3D room presentation

Before I even thought of creating a 3D room viewer presentation, I had already performed some serious model reuse, extension, and combination: a) the AOP and EDEN to construct GEL; b) GEL and the AOP to construct an HTML Environment; c) the HTML Environment with GEL to construct the Presentation Environment. Figure 4.10 illustrates the history of these activities, and my contribution to the evolution of the room models. In the last activity, I combined the Presentation Environment with the 3D room viewer to construct a 3D room presentation [Har07a] (to be further used for the computer graphics module). The following description gives some insight into the activity of combining models.

The first action to be taken was to load the Presentation Environment into `tkeden`, followed by loading the 3D room viewer model. In this state, both models were running within the same `tkeden` environment, but were not connected in any way. By introducing the definition `scoutbox_display = "screen";`, the 3D room model which is defined in screen is displayed in scoutbox which is a component within the presentation window. To be precise, the 3D room model as shown in Figure 4.6 on page 97 moved into the Presentation Environment resulting in a combined model as shown in Figure 4.13. Further redefinitions were made to resize some of the components within the presentation window,

and some other redefinitions were made to remove parts of the 3D room viewer model from the display. Apart from these, no further steps were necessary to combine the two models.

In the above combination of models there were no observable name clashes, but occasionally this is an issue that has to be dealt with. On a later occasion, in another context, the 3D room viewer was combined with the EDDI database model. In this case there was a clash of names as the 3D room viewer has a project function for transforming 3D to 2D and the EDDI model has a project function for selecting columns in a relation/table. To get around this problem, it was necessary to replace one of the functions under a new name and rename every usage of the function. It is also not obvious in `tkeden` when two observables clash from different models because redefining observables on-the-fly is a perfectly natural activity for a model-builder. This is one of the uncertain aspects of combining models, although one that 'virtual agency' (as partially implemented in current versions of `tkeden`) [Sun99] aims to address.

The next action was to create presentation slides to support a model-builder in exploring parts of the 3D room viewer model. This was achieved within the presentation window with the help of the HTML Environment and notation. Slides were created from the HTML given in Beynon's notes on using the 3D room viewer, some of which were discussed above, such as changing the 3D to 2D project function from perspective to orthogonal by redefining the eye distance. At this point I discovered that I could make use of the `%html` notation to display the state of key observables in the model. For example to display the current eye distance in a slide, the following HTML was used: `<p>The current eye distance is <eden>eye_dist</eden>.</p>`. It is worth pointing out that all of the above was achieved in a single session; at no time was there a need to close and reopen the `tkeden` environment, or recompile as you would a Java program.

In combining the models and using the Presentation Environment to create slides to support exercising the 3D room viewer, the resulting model is suitable not only for giving lectures but also for students to explore on their own or in groups. This environment is much more accessible than the 3D room viewer on its own as using the `%html` notation has allowed parts of the model to be uncovered for the learner. By highlighting key observables and possible interactions the model uncovers aspects that are relevant to learning about computer graphics. This 'making the model more accessible' has not been at the expense of flexibility as in the 3D Dino Viewer [Bha06]. The 3D room viewer model is there in its

entirety (and it is quite possible that another model-builder might use it for a different activity, such as designing a room). However, scaffolding around the model has been added that points to parts of the model that might be worth investigating for someone learning about computer graphics. The learner must still be inclined to experiment themselves, but the learning curve has been smoothed out somewhat through the addition of help with some basic experiments.

The value of combining models is that: models can be reused in different contexts (e.g. HTML Environment used for presentations); models can be extended and given greater depth (e.g. using GEL components instead of creating them from scratch); scaffolding constructed around the model can create an environment that is accessible to other model-builders (e.g. combining the Presentation Environment with a model such as the 3D room viewer to provide some support for other model-builders to explore and develop their own models).

4.2.3 Flexibility through marrying the roles of student, teacher and developer

As discussed in Chapter 3, traditional software development techniques draw a clear distinction between development and use. In terms of educational technology, the roles of student, teacher and developer are represented by activities involving use, specification and implementation respectively (see Figure 3.10 on page 71). In EM there is no distinction between development and use because all interactions with a model are deemed to be of the same nature (i.e. redefinitions invoking state-change). Hence, distinctions between activities representative of students, teachers and developers are blurred when model-building using EM (e.g. the teacher is not concerned solely with specification). The model-builder can be seen as *marrying*, *integrating*, or *unifying* the roles of student, teacher and developer.

The way in which model-building took place in combining the Presentation Environment and the 3D room viewer demonstrated more than model reuse and combination. Using the 3D room viewer for a presentation might typically be considered as a teacher activity (e.g. in preparing a lecture), but during the model-building there were other activities taking place. As I created and used the 3D room presentation, I found that there were things that the Presentation Environment could not do. So I began to make some adjustments to the underlying models. When I delved into this further, I found that some of these limitations of the Presentation Environment were due to the HTML

Environment. So I began to extend the HTML Environment also. Furthermore, I ended up changing the underlying GEL notation. What follows is a detailed account of the marrying of student, teacher and developer roles in the evolution of the 3D room graphics presentation [EMP:graphicspresHarfield2007] (as depicted in Figure 4.10 on page 103) and the implications for the flexible characteristics of learning as described in Chapter 1.

Specifically, as I began to use the 3D room presentation, I noticed that I was typing in redefinitions that were already displayed in the slides. I spent time copying what was already on the slide, occasionally making some small modifications to get different effects (e.g. changing the eye distance). I realised that I needed a way of executing the scripts and variations of the scripts that were displayed on the slide. The slides in the Presentation Environment, as it stood, lacked the potential for human interaction to be automated—it required what EDEN already has, that is agent actions (described in Chapter 2).

One of the most basic elements of agency in a webpage is the hyperlink, which usually performs an action that changes the current webpage in the browser. This action was not necessary in the Presentation Environment because the webpages had become a sequence of slides. It would be useful however, if the hyperlink could be used to trigger the execution of a script. I created an example page containing the line `<a href=writeln("hello");>click here</a>` and then I set about changing the GEL notation so that it would execute the EDEN script when the hyperlink was clicked. This required a change to the underlying `Tcl/Tk` scripts that deals with the HTML component, but this was achieved on-the-fly without restarting the model or the environment.

In the course of making this small change, I had moved from the role of student who was exercising the presentation, to the role of teacher who was creating the presentation slides, to the role of developer who was modifying the functionality in the Presentation Environment. All of these roles were played in the same environment, there was no switching of modes or views that is a common feature of many learning environments. As a learner, I had *married*, *integrated* or *unified* the roles of student, teacher and developer, as shown in Figure 3.11 on page 72.

After the above changes, it was possible to place scripts within a hyperlink on a slide that could be triggered by the model-builder. In terms of ease of use, this was an improvement, but it encouraged the 'just looking at the artefact' type of interaction that was disapproved of earlier. I was not satisfied that this exposed the model to manipulation in a flexible manner.

```
/* redefining an existing parse rule */
%aop
<html_statement2> = <html_statement3> "<page>" <html_page>
                    : do { $v is $p1 // $p2; } now
                    | <html_statement3>;

/* defining two parse rules to deal with the <script> tag */
<html_statement3> = <html_text> "<script>" <html_script>
                    : do { $v is $p1 // $p2; } now
                    | <html_text>;
<html_script> = <html_readall> "</script>" <html_statement>
              : do { $v is "format("//$p1//")//"//$p2; } now
              | <html_error2>;

/* defining a new function for displaying the script */
%eden
func format {
   para script;
   return "<pre>"//script//"</pre>"
      //"<p align=right><font size=-3>"
      //"<a href=execute(\""//script//"\");>execute</a> | "
      //"<a href=inputbox_text=\""//script//"\";>"
      //"copy to input box</a>"
      //"</font></p>";
}
```

Figure 4.14: Adding the 'script' tag and complementary HTML formatting.

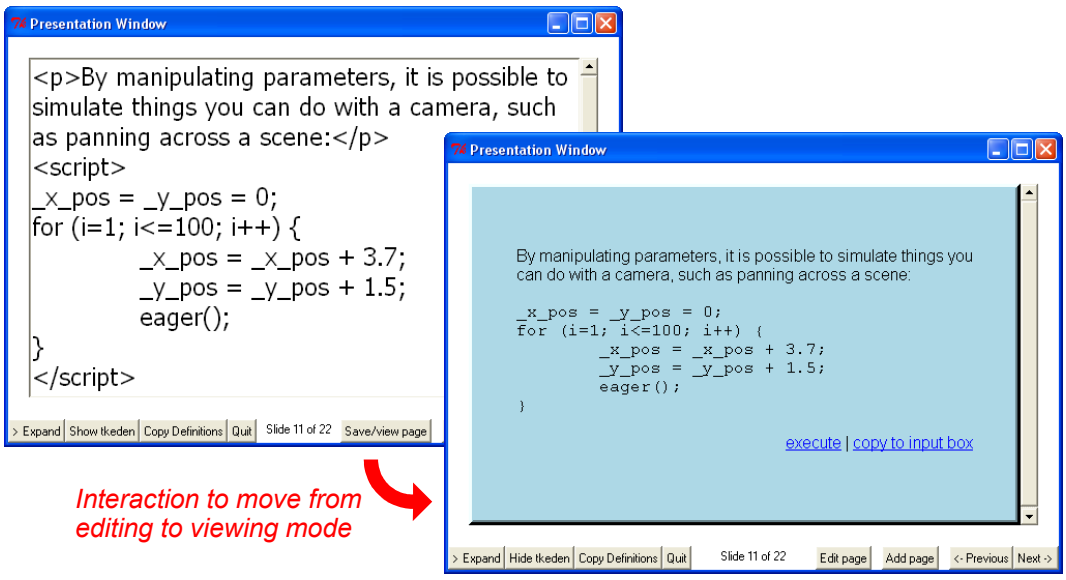


Figure 4.15: Using the 'script' tag immediately in the Presentation Environment.

The next extension I made to the environment involved adding a new tag to the `%html` notation that dealt with scripts. This new tag would contain EDEN or other scripts, and once rendered in a GEL HTML component, allow the model-builder to execute them. It would also open up the possibility of making changes to the scripts before execution by copying the to the input box. This would be a further extension of the technique I had moments ago constructed. Figure 4.14 contains the EDEN script showing how I changed the `%html` notation to parse a `<script>` tag that transformed the contained script into a formatted script with some hyperlinks for actions on the script. This required some significant changes to the `%html` notation, adding some new parse rules and changing some existing ones (see Figure 4.14). The first line of Figure 4.14 changes the existing rule to inform the AOP of the new statement, the second and third lines describe what to do with script tags, and the EDEN function converts a script into some nicely formatted HTML. After these changes to the `%html` notation, the `<script>` tag was immediately available for use in the presentation slides. Figure 4.15 shows a slide being edited to make use of the `<script>` tag and the slide being viewed with the formatted script. This second extension again shows how a model-builder, from the role of a teacher creating a presentation, can take on the role of a developer to modify the environment in which the presentation is created. The modifications are more than superficial changes such as changing the layout or design of the slides—they are fundamental changes to the notations and state of the environment. Figure 4.16 highlights the models that were involved in the interaction and shows that some interactions could be typified as teacher activities, whereas others—such as the addition of the `<script>` tag—could be seen as developer activities.

The account of EM in this section shows that a teacher creating a presentation is a student exercising the presentation for themselves, and the student or teacher can also be a developer modifying the Presentation Environment as depicted in Figure 4.16. To some extent the words ‘student’, ‘teacher’ and ‘developer’ are inappropriate to refer to each of the roles because their activities are not constrained in the same way as they are with ET as discussed in §3.3. Figure 4.16 illustrates how a model-builder is likely to play the roles of ‘student’, ‘teacher’ and ‘developer’ at different times when exercising, manipulating and constructing the model. A model-builder’s experience with the model is likely to determine the extent to which it is exercised, manipulated and constructed. As a model-builder becomes more experienced, deeper changes will no doubt be made to the underlying observables, dependencies and agency that form the model, from the

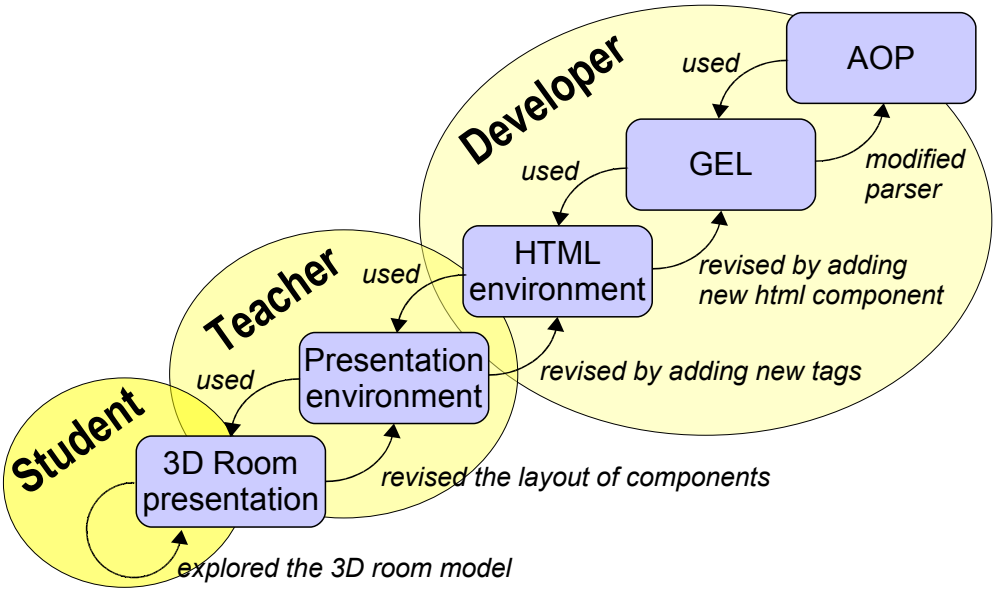


Figure 4.16: Interactions demonstrating the marrying of the roles of student, teacher and developer.

surface level of key observables in the model to a deeper level of observables uncovered from significant exploration. In this way a model-builder moves from a role of user to a role of developer within the same session and environment.

It has been shown how EM (using the `tkeden` tool) enables the model-builder to play the role not only of the student and the teacher, but also of the developer. In some respects, a learner with the characteristics set out in Chapter 1 is constantly involved in activities in each these roles; a learner studies the given material, questions and changes the way material is presented, and explores the potential for new material. Thus, an environment that enables a learner to engage with all these activities, in a continuous manner, provides the potential for significant learning. With its focus on model-builders where the roles of student, teacher and developer are married together, EM empowers the model-builder with flexibility to learn *without a prescribed outcome* and *without following a preconceived path*.

4.2.4 Implications for learning

The nature of EM activity has been demonstrated with reference to a specific topic in computer graphics. Figure 4.10 on page 103 depicts the extent to which model reuse and combination can be useful in EM. With reference to learning, model reuse and combination can lead to *flexibility* in learning on two levels. On the level of interactions, the nature of EM allows the model-builder to be concerned with observing and exploring states, not

developing and using prescribed behaviours, as differentiated in §3.1. Figure 4.16 demonstrates that a model-builder can flexibly take on the role of a teacher at one moment and of a developer at the next. In EM, exercising a model need not have a *prescribed outcome* for the learner, and exploring a model need not follow a *preconceived path*. Similarly on the level of models, the potential for model reuse and combination ensures that the possible outcomes and paths that models can take are endless. Considering the evolution of the room model in Figure 4.10, it is impossible that Yung could have envisaged the potential for teaching computer graphics, or the path that his model would follow in eventually contributing to the teaching of computer graphics. The nature of EM as explored in this section (i.e. prior to prescription and ritualisation) match the flexible strand of characteristics of learning described in Chapter 1, that is learning *occurs without a prescribed outcome* and *need not follow a preconceived path*.

4.3 Experimental learning in artificial intelligence

The final section of this investigation into particular areas of computer science education examines the experimental characteristics supported by EM in the context of learning about a specific topic in artificial intelligence (AI). As explained in §1.2.1–§1.2.3, the three characteristics that form the experimental strand of eight significant characteristics of learning are:

- learning occurs when constructing artefacts in the world;
- learning involves an active construction of understanding;
- learning results from realising the unknown.

While the previous two sections on databases and graphics demonstrate EM's suitability for teaching some standard components of a computer science course, this section examines the potential for learning in a more experimental fashion, as might be expected in research or in the primitive stages of developing teaching material. The subject of AI is chosen because a number of student projects have been undertaken relating to AI (e.g. the Wumpus model [EMP:wumpusCole2005] and the Ant Navigation model [EMP:antnavigationKeer2005] to be discussed further in Chapter 6) and my research has previously applied EM experimentally to explore AI and social behaviour in studying

human-robot interaction (HRI)[†]. Therefore, this section relates to computer science education primarily from a research perspective. However, this is not to suggest that the experimental characteristics of EM are only applicable to research—the case studies in databases and computer graphics demonstrate experimental qualities not primarily drawn on in the discussion.

4.3.1 Social behaviour, AI and human-robot interaction

Human-robot interaction (HRI) is the study of interaction between people and robots, and is concerned with developing robots that react in ways that are acceptable to people. Robots are already being used in areas such as search and rescue, bomb disposal and space exploration, and in the future it is likely that robots will be found in everyday environments. Dautenhahn, a prominent researcher in the growing field of HRI, discusses the need to consider how robots can coordinate their actions with people in a suitable manner [Dau07]. In order to develop such socially-aware robots, designers must attend to details of social behaviour—as studied by social scientists—and develop AI—as studied by computer scientists—that imitates *some* aspects of social behaviour [Dau07].

The difficulty of dealing effectively with issues relating to social intelligence in the design of robots is widely recognised. In discussing this challenge, Fong et al [FND02] identify two approaches to the design of socially intelligent agents, the 'biological' and the 'functional'. The biological approach aims to draw on understanding of animals and their behaviour to design robots which exhibit similar properties to their biological counterparts. The functional approach only takes the functionality of such robots into account and is not concerned with the mechanisms by which this is achieved. Traditional AI generally takes a functional approach. The biological approach is favoured by those interested in the social sciences and biology.

Whatever the orientation of the robot design, there are major technical and conceptual issues to be addressed in developing robots that are socially responsive. It is implausible that such issues can be resolved by implementing behaviours that are comprehensively pre-specified by abstract analysis of the operating context as would be expected with a conventional program.

There are a number of relationships that may have an impact upon social interaction.

[†]The application of EM to AI drawn on in this section is a development of ideas from a joint paper presented at the Artificial Intelligence and Simulation of Behaviour (AISB) Annual Convention, University of Herfordshire, in April 2005 [BHC05].

Three significant type of relations, as described in [BHC05], are:

- *Spatial relations*—An agent's physical location and the surrounding space are likely to affect the behaviour of the agent. Actions in small confined spaces are usually different from those in large open spaces.

- *Temporal relations*—Time plays a significant role in human behaviour. When time is at a premium humans are likely to perform tasks differently from when they have plenty of time.

- *Status relations*—The status of human agents affects their interaction and expectations. Interaction with those with whom we are familiar differs from interaction with strangers. Interaction within the working environment, families and cultural contexts is likewise differentiated according to the status of the agents with whom we are interacting.

Human beings take account of these relations by gradually learning from experience, whereas a conventional programming approach to robots views these relations as being predefined up-front. A key issue in HRI is to what extent should robots be able to learn from experience.

4.3.2 EM for learning about HRI

In the joint paper with Beynon & Chang [BHC05], it is argued that EM is a suitable approach for learning about these issues that may then inform actual robot development. The evidence is based on learning about HRI through the construction of personal models developed by myself and other model-builders. Three models are used as examples of attending to spatial, temporal and status relations.

4.3.2.1 Spatial relations

The first model of interest relates to the study of crowd behaviour and was developed by Martin as part of a final year undergraduate project. The model contains a graphical representation (2D line drawing using the DoNaLD notation [EMD]) of two people walking towards each other in a corridor—it will be referred to in this thesis as the corridor model[†]. Martin used the model for experimentation to learn about how people avoid each other

[†]Martin's project contains three variations of corridor model [Mar04:p30], but only the first and simplest will be covered here (the third and most complicated contains 25 agents).

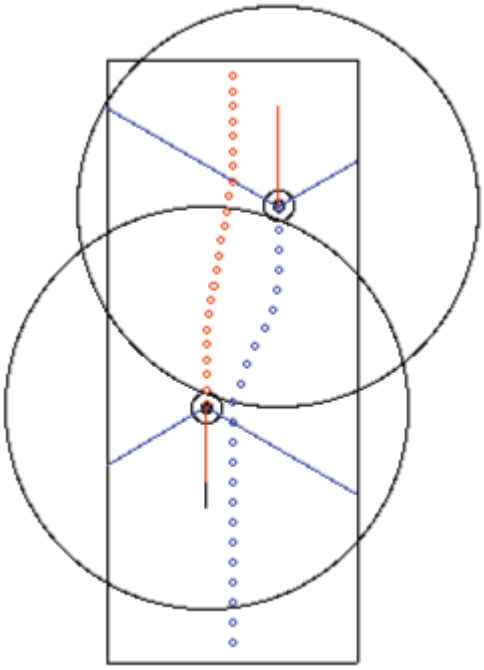


Figure 4.17: The corridor model by Martin [Mar04].

in a corridor, and therefore, it satisfies the experimental characteristic that EM can be used, in the spirit of constructionism, to construct artefacts for learning as discussed in Chapter 2.

The artefact takes account of many spatial elements: heading, direction of focus, field of attention, and personal space [Mar04:p35]. Martin experimented with different mechanisms for collision avoidance by varying the emphasis of these elements. This resulted in four types of avoidance strategy [Mar04:p37] which were compared and examined as possible strategies that could be implemented in robots. As strategies developed, observations led to changes in the model. For example, when experimenting with avoiding by focussing on a point in the distance, the agent required an observable for the re-target direction and distance [Mar04:p41]. Martin also developed an understanding of what sort of strategies for avoidance are realistically used by humans, and therefore he experimentally formulated possible theories about a specific area of human behaviour. Martin describes the benefits for learning of EM artefacts is that they are "experimental testbeds for formulating theories and possible implementations" [Mar04:p78]. This reflects Jonassen's idea of learners as theory builders where building theories contributes at a fundamental level to conceptual change [Jon06].

Martin says that model-building "stimulates ideas and discussion" [Mar04:p80]. Many of the issues that his model-building brought up related to spatial relations. Examples

given include the introduction of a stairway into the corridor model that could have ramifications for the agent's field of vision, a doorway that might force agents to stop and wait or even hold the door for another, or how an agent walking side-by-side with a stranger might change their pace to avoid uncomfortable synchronisation.

Learning occurs when constructing artefacts in the world because construction leads to experiences on all levels of the EFL (see §2.2.8). By constructing models, learners have to think about the formal definition of a phenomena, and by using artefacts there is a practical experience of the phenomena. In everyday life we have the concrete experiences but not the abstract ideas about how we avoid people in a corridor (i.e. we don't think about it, we just do it). At the other extreme there is modelling in mathematics which may force us to think in great detail about the formal definition of a phenomena but often without being able to test the abstract constructions in concrete situations. The benefit of model-building, specifically with reference to observation, dependency and agency, is that we can construct models and experience them. For example, in the corridor model there are two agents walking towards each other, one is played by the model-builder and the other is played by a semi-automated agent. On the one hand this represents a concrete situation that a learner studying HRI could relate to and participate in by playing the part of the human walker. On the other hand there is the semi-automated agent which the model-builder can use to experiment with possible avoidance strategies based on experiences in the model and in the world.

By comparing interactions with the artefact to interactions with the referent it is common that there will be discrepancies or situations where behaviour in a corridor may deviate from the normal. If the person walking towards you is holding a white stick then you might take extra care in moving out of the way. Or if moving through a doorway and the person walking towards you appears to be elderly then a respectful action would be to hold the door so that they are inconvenienced as little as possible. The fluid nature of EM activity means that the personal, particular or subtle needs can incorporated in the stream of model-building—this is a significant characteristic of EM that is beneficial in learning.

4.3.2.2 Temporal relations

The emergency egress model was constructed by Howard Perrin as part of a submission to WEB-EM-2 [WEBEM2:Emergency Egress Simulation]. It was inspired by Martin's corri-

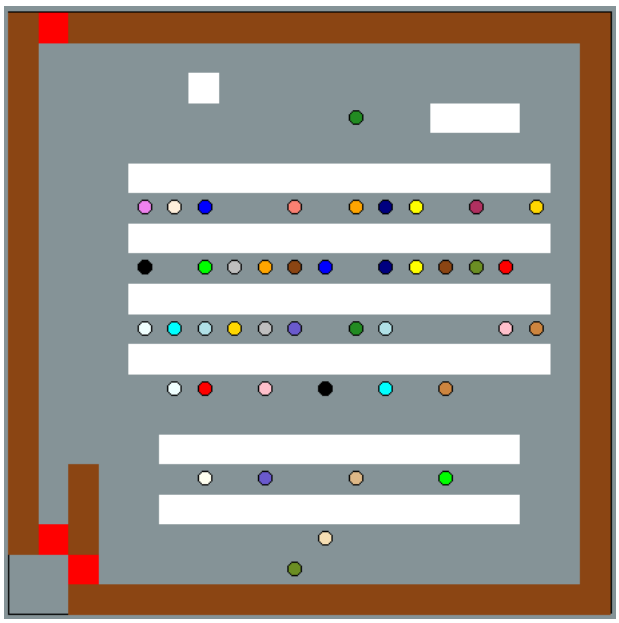


Figure 4.18: The emergency egress model by Perrin [WEBEM2:Emergency Egress Simulation].

dor model and builds on Martin's initial contribution to understanding crowd behaviour. In particular, the model considers the situation of a large room (e.g. seminar room) or building (e.g. Department of Computer Science) containing many agents (e.g. students) and the event of a fire alarm.

The model includes many spatial aspects similar to the corridor model: people in the model have a physical or personal space that cannot be penetrated by other people, people avoid walking into walls and cannot see around or through walls, people cannot walk over desks and other obstacles, although they can see around them. Spatial relations might change depending on the status or time (e.g. during a fire alarm it might be that people are quite comfortable moving through a doorway more than one at a time).

The main temporal relation in this model involves the fire alarm. When the time comes for an emergency exit, the agents in the model all change their focus to exiting from the room. This might involve searching for an emergency exit sign or door, observing other agents head towards an exit point, moving to an exit, or queuing to get to an exit.

The emergency egress model can be used to learn about the design of rooms in terms of where to place exits and furniture, or to test out emergency situations to find out the time required to vacate a building. Relevant to learning in this section is the idea that such models can be used to learn about human behaviour so that it is possible to build robots that exhibit similar characteristics to people.

One of the key elements to constructing models that mimic human behaviour is to

construct the model from the viewpoint of the person or agent inside the model. In the emergency exit situation, the agent is unlikely to have a complete knowledge of the layout of the building including all the fire exits or a precise knowledge of the closest fire exit from any given point in the building. A computer model may be useful for calculating these things, but they are not necessary for successful evacuation. It is possible that a person in a familiar building might think through the best way to escape from the building in an emergency, but sometimes all that is needed is a simple strategy like 'follow your neighbour'. By the model-building putting themselves in the position of someone trying to exit a room in an emergency is useful because not only might it lead to more realistic agents, but also it connects practical knowledge of emergency situations with formal knowledge of strategies for evacuation. Such an approach to model-building can lead to insights as was discovered in the emergency egress model where Perrin began with a simple model that involved an agent walking towards any exit. With different room designs it was found that there are often obstacles to moving directly to the exit, therefore an agent must walk around desks and chairs in order to get to the exit. Next Perrin assumed that some agents might not know where the exit is in the room, and such agents would have to look for an exit—and if they could not observe an exit then they would have to go searching for one.

By actively constructing the model in response experiments with different room designs and different agent strategies, Perrin came to an understanding of the issues involved in developing a more formal knowledge of emergency egress—particularly from the viewpoint of an agent. Such knowledge is useful when considering HRI, but not necessarily in terms of implementing robots that are able to successfully vacate a building in an emergency. More important to HRI is the temporal relations that may impact on social interaction. In an emergency it is important that the robot does not hinder the evacuation process, therefore it might be useful for the robot to have a model of human behaviour in order for it to avoid obstructing people exiting the building—it is probably not desirable for the robot to join in the rush for the exit. As time progresses, it might be suitable for the robot to exit the building, or for it to take on other responsibilities such as checking the building for stranded people. Perrin's model can take account of such situations as he explains, by for example, making an agent move very slowly to simulate a person with a broken leg. A possible next experiment for a model-builder interested in HRI would be to see if one of the agents could act as an observer looking for peculiarities such as slow

moving people—such agents may be useful in an emergency situation[†].

In order to explore such relationships in HRI, the EM approach to model-building engages the learner in active construction. EM's approach of identifying patterns of observables, dependencies and agency enables a close relationship between experiences in the world and in the model. Perrin's model is connected to issues in crowd behaviour that Martin was investigating in the corridor model, but Perrin started from scratch in the construction of his model. In this way, experimentation was not constrained to exercising an existing model—like experimentation in microworlds according to Jonassen [Jon06]—but actually involved constructing a model to develop some understanding. Therefore the emergency egress model shows support for the characteristic 'learning involves an active construction of understanding' described in §1.2.2. A key factor in EM's support for active construction is the well-conceived foundation of modelling with dependency as described in §2.2.2.

Further refinements relating to temporal aspects of emergencies might involve a 'panicky' aspect, whereby the longer a person has been waiting to escape the room the more they become inpatient and the less likely they are to act in an orderly fashion. In such cases observing social distances will be less of a priority than getting through the fire exit as quickly as possible. It is through experimentation with observables, dependencies and agency that the model-builder will be able to recognise and address subtle features of the problems in HRI.

4.3.2.3 Status relations

The first objective in applying EM to HRI would be to better understand how human capabilities and behaviours and robot capabilities can be most effectively elaborated and integrated. As illustrated in the corridor and emergency egress models, EM can help explore the factors that are significant in determining human behaviour in relation to such tasks as collision avoidance. It can also enable the construction of prototype behaviours expressed in observables, dependencies and agency that serve as useful models for devising and analysing robot behaviour. A more ambitious goal involves demonstrating that EM can used in programming robots, especially where low-level interface issues such as image recognition are a concern. Such questions are not elaborated in this section where the focus is on learning about HRI—for a detailed discussion of the application of EM to HRI

[†]A detailed description of HRI issues in urban search and rescue is given by Murphy [Mur04].

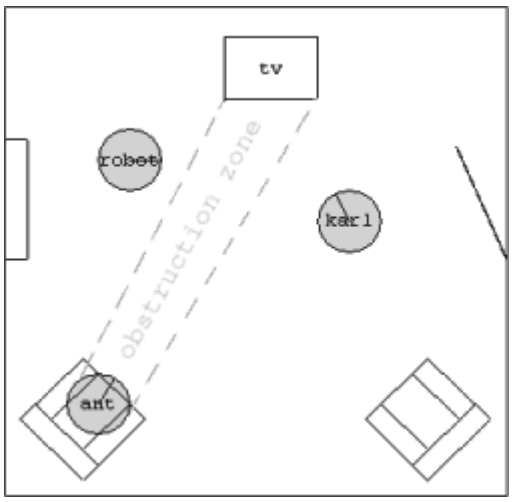


Figure 4.19: The living room in the house robot model.

refer to Beynon et al [BHC05].

The third model to be considered explores the situation of a domestic robot interacting with people in a house. It could serve as a useful model for learning about the issues in HRI, and particularly for experimenting with various configurations of observables, dependencies and agency for robots. The model uses elements from a model developed by Sunny Chang in her final year undergraduate project in modelling an intelligent house environment [Cha04]. The house robot model that I created in `tkeden` was used to experiment with various familiar situations in the house where a robot might feature or be considered useful in the future[†].

Dependency plays a key role in all forms of HRI and experimentation with dependencies can inform possible robot behaviour. By way of illustration, consider the dependency involved in a living room scenario where there are people watching television, as depicted in the screenshot shown in Figure 4.19 taken from the house robot model. Clearly it would be undesirable for a house robot to obstruct someone while they are watching television. In a similar way to the use of dependency to detect collisions in the corridor model, a dependency (or set of dependencies) can represent a potential obstruction by the robot. Figure 4.19 designates a potential obstruction zone for the robot. Using dependency, these areas can be defined in terms of the position of other agents and the television, so that if a person moves then the robots awareness of an obstruction is updated. Another issue that should be considered is if there is actually an obstruction. The *status* of the television also determines whether there is an obstruction—if the television is switched off then the robot

[†]It has been suggested that most people like the idea of robots taking over the menial jobs around the house—whether it is possible, and when it is likely to happen, is yet to be seen.

```
tv_is_on = true
tv_position = {16,28}
tv_left = tv_position - {3,0}
tv_right = tv_position + {3,0}

chair1_position = {5,5}
chair1_occupied = true
chair2_position = {27,5}
chair2_occupied = false

robot_position = {10,20}

robot_is_obstructing_chair1 =
      tv_is_on &&
      chair1_occupied &&
      inside_triangle( robot_position,
      chair1_position, tv_left, tv_right )

robot_is_obstructing =
      robot_is_obstructing_chair1 ||
      robot_is_obstructing_chair2
```

Figure 4.20: An extract from the house robot model showing that an obstruction is dependent on the position and status of other agents.

can be fairly sure that it is not being watched. The obstruction is then dependent on: the robot being inside the area between the person and the extremities of the television, and the television being switched on. Thus, there are *spatial* and *status* relations to be consider in this situation, as well as a *temporal* aspect in that the status and position of the agents may change over time. The way in which these dependencies can be directly manipulated in `tkeden` is illustrated in a small extract of script in Figure 4.20, selected for this example from the house robot model shown in Figure 4.19. Further dependencies could be introduced at any time, for example some people might be particularly sensitive about how much space is occupied around or behind the television—therefore the dependencies relating to `robot_is_obstructing` could be modified so that the robot is deemed to be obstructing if it is behind the television.

The house robot model can support learning about HRI due to the focus on experimenting with a situation in which we are generally familiar. By taking a familiar scenario such as the living room of a house and introducing a new agent (e.g. a robot agent) into the picture we are 'building up' the model incrementally, situating the unfamiliar new element in a familiar environment, from the perspective of an external observer and also from the perspective of the new agent. Model-building is not a one-off exercise, as is typical in programming to create a complete program with specific outputs; it is focussed on building up or evolving a model. The activity of model-building establishes—through experimentation—an intimate relation between the artefact itself and the mental model

of the model-builder. Thus, the activity is a support for learning. Experimentation is the key to the evolution of this relationship, and very much involves considering and exercising familiar aspects of the artefact in order to discover what is not familiar in our mental models.

Consider a possible evolution of the house robot model, in a larger living room scenario with more agents, where the robot is given the task of moving from one side of the room to the other—in order to reach the cocktail cabinet to replenish it's drinks tray for example. The robot now has to combine the avoidance of obstruction zones with the avoidance of people moving around the room whilst completing a task. One way to build up the model would be to combine it with the corridor model, and experiment with the effects of this conjunction. That modelling social interaction should be evolutionary and open-ended is completely plausible. As we do not fully understand the nature of social conventions [Gil95]—even our own—it is unlikely that we will ever want to finalise (or completely formalise) a behavioural model.

In another experimentation session I considered how a robot might learn or adapt its dependencies associated with how much space a person is comfortable with before an obstruction or intrusion is felt. When two people are talking together they assume a comfortable distance with which to communicate, when standing in a queue people have a natural feeling for how close they should stand, when walking past someone it is not considered natural to brush shoulders or to take particularly evasive action to avoid the person. In *The Hidden Dimension*, Edward Hall devotes an entire book to these topics and he describes in detail the subjective personal spaces that people keep with one another [Hal66]. From Hall's descriptions, and from personal inspection, it is obvious that such personal spaces for an individual develop over time from experiences, and cannot be formalised even within a particular culture. My concerns were related to how we might become aware of personal spaces and learn to respect them, from the position of an agent like the house robot. My strategy involved giving the robot agent a rough distance to keep between itself and other agents and then modifying this when the robot noticed that other agents moved away when close to them. Experimenting with this model led me to realise that there are many other issues that were initially unknown to me. For example, if a person walks towards the robot there might be expectation that the robot would move out of the way, and not necessarily adapt to a new closer social distance.

By exploring these HRI issues using the house robot model, EM's potential for sup-

porting experimentation in learning has been demonstrated. Experimentation is an incremental activity which involves building models of what is familiar and using them to highlight the unfamiliar—it is a particularly fluid activity to which EM is well-suited. As models are built up, the model-builder becomes aware of relevant 'residue' issues and can begin to experiment with patterns of ODA to develop a deeper understanding of the issues. In this way, EM offers support for *learning resulting from realising the unknown* as characterised in §1.2.3.

4.3.3 Implications for learning

The EM approach offers potential for learning about issues in HRI by building on previous work [BHC05]. The above discussion highlights the importance of EM's support for the experimental characteristics of learning as introduced in Chapter 1. The introduction to EM in §2.1 highlights the nature of EM construals that in some sense 'embody knowledge' [BHC05]. Construals are particularly connected to personal interests, situations and experience, and this makes them ideal for experimentation that involves a necessary human element [BR07].

The construction of construals using EM tools is an active process of incrementally building up and continually refining the artefact in response to experimentation. As highlighted by the house robot model, experimentation and construction are open-ended—it is the model-builder who actively chooses in which direction to explore. Artefacts start simple: observables and dependencies are created from scratch, or an existing model forms the basis of further model-building. In the context of HRI, as spatial, temporal or other relations become apparent, observables and dependencies are elaborated that embody more complex ideas. Where interactions and observations with the model do not reliably correspond to interactions and observations with the referent, the model-builder engages in an experimental sense-making activity in order to develop a deeper understanding of the model and the referent. Experimentation is the key activity that enables a correspondence to develop between the model and the referent as depicted in Figure 2.5 on page 39. Experimentation promotes the development of understanding and plays a significant role in learning about specific topics in HRI, graphics and databases as demonstrated in this chapter.