Higher-order Constructs for Interactive Graphics and Design in a Definitive Programming Framework

Dominic Gehring†
Simon Yung†
Richard Cartwright†
Meurig Beynon†
Alan Cartwright‡

†Department of Computer Science, University of Warwick, Coventry, CV4 7AL
‡Department of Engineering, University of Warwick, Coventry, CV4 7AL

1 Introduction

Systems of definitions to represent dependencies between graphical elements were first exploited by Brian and Geoff Wyvill many years ago (see e.g. [22]). Similar principles have been applied to knowledge representation in CAD support systems [19] and advanced geometric modelling applications such as scientific visualisation [20]. Our well-established programme of research at Warwick, *The Empirical Modelling Project*, has involved an in-depth study of the application of such definitive (definition-based) principles in modelling and programming with particular reference to interactive graphics and CAD [1, 3, 4, 8, 9]. The recent release of a public-domain version of our **tkeden** interpreter represents a milestone in our project, and a crucial stage in our development of these principles. Within the tkeden environment, there are rich possibilities for interacting with definitive scripts through redefinition of variables, and for constructing systems of interacting agents. The possibilities for generating and managing scripts and agents within the system are on the other hand more limited, and are represented by a range of features that are not well integrated into the framework of our current abstract computational model the Abstract Definitive Machine [5]. Informally, our exploitation of definitive scripts and agency is inhibited by a relative lack of support for forms of hierarchical abstraction.

This paper gives a brief review of the current status of our Empirical Modelling principles and tools, and explains how these can be extended to accommodate higher-order observations and definitions. Some examples are used to illustrate how higher-order definitions can be exploited in graphics and animation. The implications of higher-order abstractions for interaction and agency, their significance in the Empirical Modelling process, and the prospects for effective implementation in a novel architecture are also discussed.

2 Background

In our modelling method we use definitive scripts to represent real-world systems. We begin by identifying things to observe in the system. In a definitive script, a variable is introduced to represent each of these observables. A variable can either be given an explicit value or defined in terms of other variables. The definition of a variable corresponds to establishing a relationship between that observable and those referenced in its definition. These relationships form uni-directional dependencies. If a variable is changed then the state is maintained; as in a spreadsheet these changes are propagated throughout the state along the dependencies that have been established. In this manner we create a metaphorical representation of an external state. Dependencies between observables express the way in which an action that changes the value of one variable indivisibly affects the values of others, and thereby delineates the immediate extent of our actions. In general, the changes to the system state cannot be attributed to a single source. For this reason, it is useful to identify many agents that can act within the state. We specify the characteristics of such agents in the specially developed LSD notation, in which we declare what agents can observe (oracles), what they can change (handles) and their protocol for doing so [6].

Definitive scripts together with agency are a powerful mechanism for interaction [1], animation [7, 12], and for constructing cognitive artefacts in design [9]. A cognitive artefact directly represents some real-world system at an appropriate level of abstraction. Such a representation can be incomplete and uncircumscribed, which is especially useful in the simulation of dynamic or reactive systems. Through a process of experimentation the model can be refined until the required behaviour is achieved. In such a context the use of graphics is essential to communicate the state of the model.

Tkeden is a general-purpose interpreter for a family of definitive notations dealing with line-drawing, window layout and various traditional types of data. Construction of models in the tkeden environment is characterised by a most unusual degree of flexibility in interaction. This lends an exploratory character that favours design, requirements modelling and experimentation. In the present stage of development of tkeden, this very flexibility inhibits higher-level structuring of observables, so that there is little built-in support for hierarchical abstraction involving scripts or agents. The lack of higher-order constructs has two consequences. Firstly, the interaction is often too low-level to reflect the constraints on real world observables. Secondly, as a process of design or exploration enters the specification phase, and flexibility becomes less important, we lack the means to express commitments, impose structure and circumscribe behaviour.

Tkeden is typically used as a stand-alone system. To some extent, the problems of imposing structure can be addressed by adding a front-end. These include translators for the Abstract Definitive Machine and for more sophisticated definitive notations (such as ARCA [2] for combinatorial modelling and CADNO for geometric modelling [8]). The technical difficulties of integrating such tools into tkeden are symptomatic of the problem of assimilating higher-order abstractions in a principled manner. Many of our motivating examples and ideas arise from the attempt to meet this challenge.

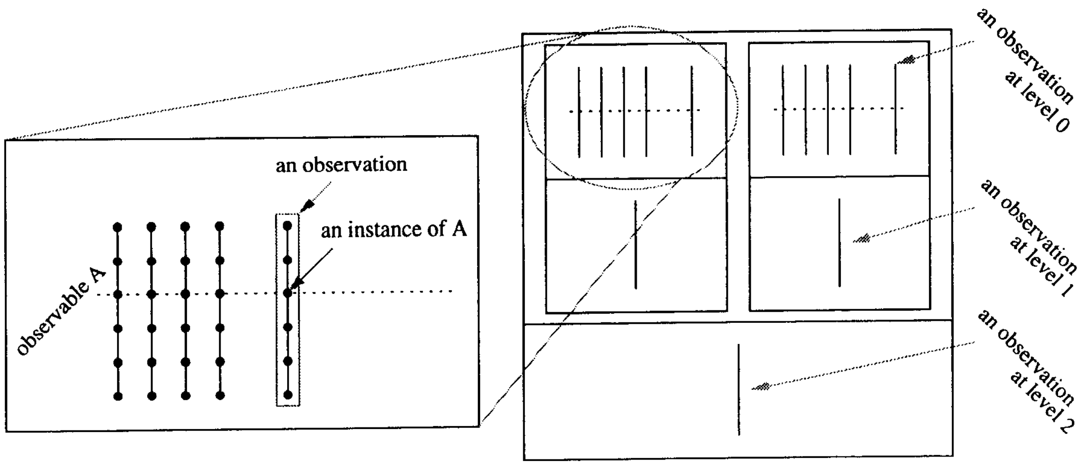

an observation
an instance of A
observable A
an observation at level 0
an observation at level 1
an observation at level 2

Figure 1: Levels of Observation

3 Higher-order Observables

The elaboration of our Empirical Modelling method can be best described with reference to a concept of higher-order observable that has been implicit in our previous work. Giving a complete account of this concept will be one of the major concerns of future work. For the present purpose, it is enough to give an informal indication of how higher-order observables are defined, and what range of applications can be envisaged.

Our modelling method is aimed at constructing a computer model whose state metaphorically represents the state of an external real-world system. The variables in our models, like the cells of a spreadsheet, typically represent specific observables. When we ascribe definitions to these variables in a definitive script, we express the way in which we expect a change in the value to a particular observable to propagate changes to other variables in an indivisible manner. The identification of indivisible relationships requires knowledge of system behaviour that is acquired experimentally, and in general involves commitment to a global assertion about how changes to observables are synchronised in all system states on the basis of partial exploration. For systems of any complexity, it can be difficult to make global assertions about state: it is more realistic to isolate particular phenomena within the system, defined by a particular selection of the observables, and analyse these in isolation. The description of system behaviour on the basis of experimental knowledge of subphenomena is a characteristic ingredient of engineering design and of the scientific method.

Figure 1 symbolically depicts the framework within which our current modelling techniques operate. Each bold vertical line on the left represents a system of values assigned to variables to represent a particular observation of a family of real-world observables. Such an observation is conceptually a snapshot of a phenomenon, in which the values of all observables have been recorded in one and the same context, as if at the same instant. Note that this concept of instantaneous observation does not necessarily presume that the current time is observed. For instance, in validating Hookes' Law, we neither need to record the time at which an observation is made, nor to record the extension of the string and the load at the same instant. On the other hand, to discuss phenomena in which time is of the essence (such as control system dynamics), the time on a clock will be one of the observables.

The collection of solid lines corresponds to a set of distinct observations associated with a phenomenon. The possible interpretations of this set are rich and varied. Significant issues are:

- to what extent is the choice of observations under the observer's control? Is the choice associated with a strategy for observation?
- is the set of observations circumscribed? For instance, is it known to be a finite set, or to be defined by a system of primitive transformations?
- is the collection of observations ordered or not? In particular, does each include the time of observation?

As in traditional scientific experiment, the most appropriate selection of observables and the characterisation of the contexts for observation is an exploratory process. Empirical judgement is likewise involved in determining whether a particular observation is admissible. Subject to making this characterisation of the family of admissible observations, we can then regard any property that is associated with such a family of observations as defining a higher-order observable. Significant examples of such properties not only include indivisible relationships, but conventional logical constraints and structural relationships between variables. In this section, we will confine our attention to indivisible relationships as higher-order observables, with a view to motivating the concept of higher-order definition.

The status of these concepts is symbolically depicted in the right-hand section of Figure 1. At *level 0* in the observation hierarchy are the raw observations of a phenomenon, as defined by explicit values of observables. At *level 1* are the indivisible relationships between observables that are expressed in a conventional definitive script. Notice that in this correspondence between *level 0* and *level 1* there may not be a direct 1-1 mapping between observables at *level 0*. This possibility arises because several observables may have to be perceived as belonging to a single structure. For instance, the *level 1* definitive script specified by variables n and l where n is a positive integer and l is the list defined by l=[1,2,...,n], can be regarded as describing a phenomenon which is represented by n+1 integer variables, $n, l_1, l_2, ..., l_n$.

To generalise to higher-order definitions it remains to note that many instances of what can be regarded as the same phenomenon may arise within a complex system. In such a case, there may be indivisible relationships between the higher-order observables associated with instances of a phenomenon are present in a system. As indicated symbolically in Figure 1, properties that refer to relationships between *level 1* observables can be construed as *level 2* observables at a yet higher-level of abstraction. Informally, the process of building up higher-level abstractions can be viewed as identifying patterns between values (this is a *level 1* observation), then patterns between patterns between values (this is a *level 2* observation). Some examples to illustrate applications of such abstractions in geometric modelling and animation follow.

4 Illustrative Examples

In this section we introduce three examples of problems, arising from our research into geometric modelling and animation, that require higher-order definitions. Three kinds

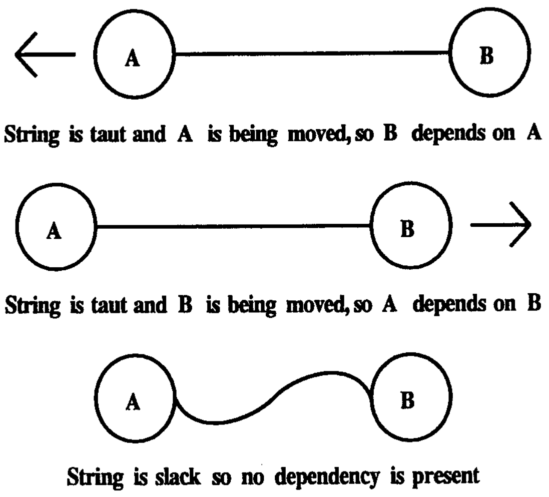

A
B
String is taut and A is being moved, so B depends on A
A
B
String is taut and B is being moved, so A depends on B
A
B
String is slack so no dependency is present

Figure 2: Blocks Connected by String

of higher-order definition will be considered: case, similar and generic definitions. We shall not develop the whole theory and syntax here, but show the principles.

Many phenomena are most naturally expressed in terms of constraints, rather than uni-directional dependencies. To simulate interaction with a complex object, as in moving a table, we need to take account of the way in which the dependency between elements of the table is determined by where force is applied. The networks of dependencies that arise in constraint satisfaction systems are designed to address these problems, but these cannot be readily expressed using definitive scripts alone even when making use of context dependent (e.g. "`if ... then ... else ...`") definitions, since such scripts appear syntactically to invoke cyclic definition. Similar considerations apply when modelling the movement of a person from the environment of one room to another, where the entire environment of observables changes as a result of a relatively primitive action.

The application of higher-order definitions to problems of this nature can be illustrated in animating a simple system consisting of two blocks connected by a string moving under the concurrent control of independent agents [4, 15, 16]. In such a system it is appropriate to reconfigure the dependency relationship between the positions of the two blocks dynamically as the string changes between loose and taut states (Figure 2). In our present framework, such dynamic constraints are difficult to represent effectively using definitive scripts, even when they are used in conjunction with agents. The problem lies in the way in which motion of the blocks (represented by a redefinition of the location of the block within the context of a script representing the dependency between block locations) directly affects the dependency relation (and so must reconfigure the script).

Using our framework for observation we can record the dependency relations that hold in different cases, through *level 1* observations (i.e. the dependencies shown in Figure 2). Observing the system from a higher-level we can identify the conditions that determine each of these cases (i.e. the conditions in Figure 2). That is, we can identify what the dependencies depend on; a *level 2* observation. In our modelling method this corresponds to a case definition at *level 2* (for each of A and B), which evaluates to give the current dependency definition at *level 1*, which in turn evaluates to give the current value of position for each block, at *level 0*. A feature of this approach is the ability to

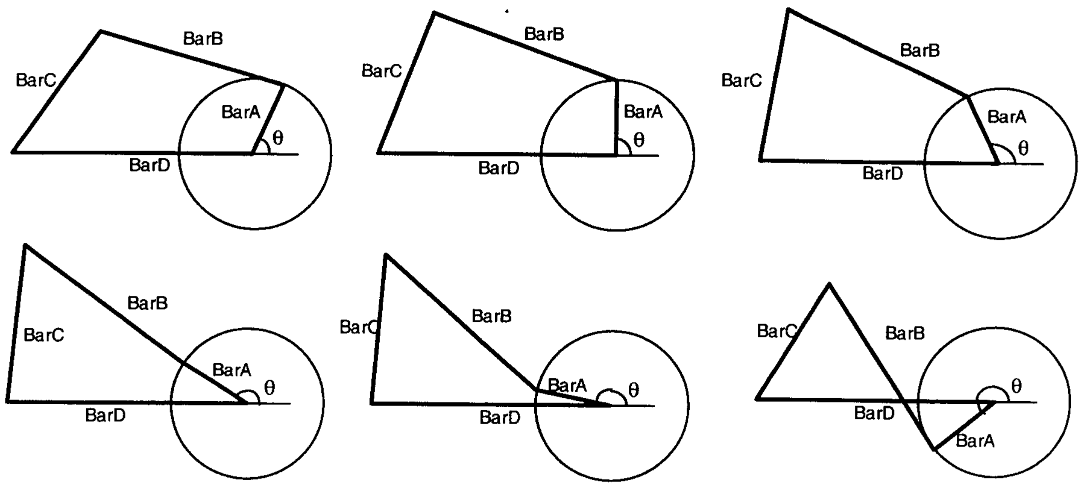


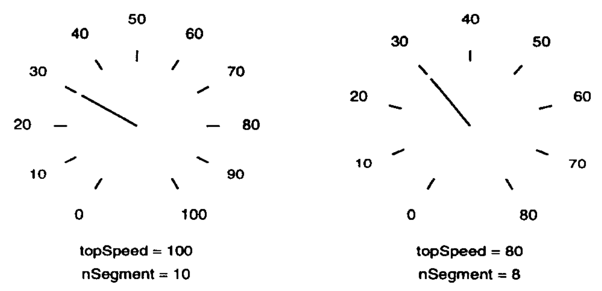
Figure 3: Four Bar Linkage

introduce forms of feedback that do not represent cyclic definition.

In the previous example a small repertoire of cases could be identified in the dependency structure. This is analogous to a variable attaining one of a small set of values, depending on a condition. Sometimes we can observe more uniformly defined patterns in the dependency structure, analogous to a variable getting its value from the evaluation of a non-singular function. In such a system we would make the *level 2* observation that a certain dependency X is always similar to another dependency Y, in that when Y changes X changes accordingly. This concept of similarity is represented in our Table notation [14] - a generalised form of spreadsheet - by a persistent form of the traditional operation of copying a formula from one cell to another (cf. the linked rather than embedded copy in Windows [18]). That is, we use similarity to establish a *level 2* definition f_Y = $\texttt{similar}(f_X)$ that evaluates to the *level 1* definition of the formula (but with the cells referenced changed relative to position), which in turn evaluates to give a value. If the defining formula f_X of X were to change, our definition at *level 1* would then change accordingly. In this way we can construct dependencies between dependencies.

This principle is extremely useful in modelling and design, for example in the specification of a four-bar linkage that has to be depicted in several different configurations (cf. [11]). This is achieved by specifying a set of *level 1* definitions for the linkage. The configuration is characterised by an angle theta and a location. Other instances can be defined at *level 2* to be similar to the first set of definitions but with different variables representing the angle and the location. This kind of *level 2* definition resembles a persistent form of the range copy operation in a spreadsheet. Any changes to the first linkage, such as altering the length of one of the bars will be propagated to the others, however they will have their own local angles and will therefore represent a different configuration of the linkage (Figure 3).

In the process of design it is often desirable to specify instances of a generic object. For example, the designer of a speedometer would wish each of its calibrations to be similar and to come from one generic specification (cf. Figure 4); perhaps such that the number of calibrations is related to the vehicle's top speed [6]. In such a system the specification of the calibrations operates at a higher-level of abstraction (*level 2*) than the definitions of the lines that make up a single calibration (*level 1*): indeed, the



```
graph speedo
within speedo {
   real needleLength = 100.0
   real minA = 4 * pi div 3
   real maxA = - pi div 3
   real A = minA + (maxA - minA) * ~/curSpeed div ~/topSpeed
   line needle = [{0,0}, {needleLength @ A}]
   real gap1, gap2, LSpc
   gap1, gap2, LSpc = 10.0, 30.0, 50.0

   x<i> = ~/topSpeed * <i> div nSegment
   f<i> = minA + (maxA - minA) * <i> div nSegment
   nSegment = 8
   node = [
      label: label(itos(trunc(x<i>)), {(needleLength + gap2 + LSpc) @ f<i>});
      line: [{(needleLength + gap2) @ f<i>}, {(needleLength + gap1) @ f<i>}]
   ]
   segment = []
}
```

Figure 4: Speedometers and Definitions

number of calibrations (and hence the number of definitions of lines) is dependent on this specification. This is analogous to defining a list of n values, because it groups together many observables into a single higher-level observable. In the speedometer example we would have a single template in the form of a *level 2* definition, that would evaluate to many definitions of calibrations at *level 1*.

Limited support for generic objects is provided by the graph abstraction in tkeden. This construct makes it possible to reconfigure the speedometer in Figure 4 by redefining the top speed of the vehicle. An alternative way to specify a family of similar values is to use a locus construct, which creates a comprehension by recording a history of values. Related abstractions have been explored in the notation EdenCAD, a variant of the tkeden interpreter that was developed by Alan Cartwright for CAD applications [11]. EdenCAD is written in AutoLisp within the AutoCAD environment. The use of Lisp as the implementation language has certain advantages in respect of higher-order abstraction, and simplifies the task of generating variable references.

5 Interaction and Agency

Our examples illustrate how the use of higher-order abstractions simplifies the specification of agent interaction, especially in contexts where this interaction is to be constrained to reflect empirical knowledge. Higher-order constructs enable us to express the way in which actions directly affect dependency (cf. the blocks), to describe generic objects (cf. the speedometer) and to make comprehensions of state (cf. the locus of a linkage).

Other informal applications of higher-order concepts are implicit in many aspects of our previous work. In effect, *level 2* abstractions are associated with functions that return scripts. A typical interpretation of a script is as a system of indivisible relationships expressing dependencies, but scripts can also serve to represent:

- roles of an agent,
- instances of a generic agent,
- alternative visualisations of a data set.

The use of scripts to represent different roles of an agent has been illustrated in connection with a case-study based on Harel's statechart for a digital watch [9], in which the substates of the display state correspond to different display functions of the watch. Instances of a generic agent can be represented by a parametrised entity in the Abstract Definitive Machine [5]. The use of a variety of definitive scripts to provide alternative parallel visualisations is a theme that has been illustrated in a variety of case studies, such as that concerned with lathe shaft design [1].

Providing more formal support for definitive scripts at a higher level of abstraction has implications for the specification and maintenance of dependencies. Issues to be addressed include:

- how to specify higher-order dependencies conveniently and clearly;
- how to ensure that higher-order dependencies are respected once established.

At present, our use of higher-order definitions involves relatively inelegant syntactic constructions based on macro expansions (e.g. in specifying analogue variables and the generic speedometer) or complex conditional definitions (e.g. blocks). The Table notation [14], currently under development, adds a table data type to tkeden and is intended to allow the user to specify higher-order relationships between tables through a graphical user interface. This adds functionality to a conventional spreadsheet, making it possible to link the definition of a cell to the definition of another, as illustrated above. By such means, it is possible to describe a parametrised table that can either be a multiplication table or a segment of Pascal's triangle. When used in conjunction with definitive notations for graphics, such structures potentially admit geometric application to problems such as the specification of fractal objects.

The use of higher-order abstractions to specify structure is subject to a general principle that what has been defined at a high level of abstraction cannot be piecemeal redefined at lower levels of abstraction. For instance, when an instance of the generic speedometer has been defined, it is not appropriate to subseqently modify its components if we wish to maintain the high-level dependency of the visualisation script upon the numerical parameter `topSpeed`. This form of higher-order definition, in which a numerical parameter determines the structure of a script, can however be exploited as a generator for a script that can subsequently be used freely for interaction at lower-levels of abstraction. In effect, the dependency between the script and the `topSpeed` parameter is removed by partial evaluation, leaving a speedometer definition that uses only conventional *level 1* definitive variables.

In general, it is essential to be able to use different levels of abstraction even in specifying a single geometric object. The definitive notation ARCA, developed with the specification of combinatorial graphs such as Cayley diagrams in mind has a sophisticated generic solution to the problems of disciplining interaction to eliminate conflict between definition at high and low levels of abstraction. The fundamental ARCA data type, the diagram, has a complex hierarchical structure comprising sets of vertices and colours, where each vertex is a coordinate vector, and each colour is a (partial) permutation of vertex indices representing a family of directed edges of a particular colour. The mode in which a variable of type diagram is to be defined is declared using an auxiliary definitive notation. This mode then serves as a template for subsequent definition and redefinition of the variable. In particular, the mode of a variable determines whether it is to be defined as a collection of subcomponents, and whether its components are to be treated as independent variables for dependency checking. For instance, the mode definition `mode` v `= abstract vertex` admits the abstract definition of v by any formula that returns a vector, such as $v = w + [1,0,1]$ but excludes the definition $v[1] = 3$. It also triggers a re-evaluation of the variable $x = v[2]$ whenever the value of v is changed in any way. In contrast, the mode definition `mode` v `= vertex 2` allows the components of v to be independently specified, as in v`[1] = 2 *` v`[2] ;` v`[2] = 3`, and (in the context of these definitions) will not trigger the re-evaluation of the variable $x = v[2]$ when the new assignment $v[1] = 2$ is made.

6 The Empirical Modelling Design Process

In our Empirical Modelling process, we aspire to comprehend the states of a phenomenon in terms of higher-order observables. Typical examples of such observables that in general exist at every level of abstraction include:

- indivisible relationships *represented by* definitive scripts,
- constraints *represented by* logical relationships,
- associations of observables *represented by* data structures.

Constraints can be made explicit either as imposed conditions that cannot be violated in any state transition, or as monitored conditions whose violation is reported. They can also be accumulated as ingredients of a formal specification. As examples such as the generic speedometer illustrate, apprehending structure is an essential component in identifying and expressing indivisible relationships at higher levels of abstraction.

In practice, the partial knowledge that contributes to comprehension of state has to be represented by using the computer as an artefact whose state directly represents the experimental contexts in which higher-order observables are identified, via an appropriate metaphor. In this aspect, the modelling process relies essentially upon perceptualisation, and on visualisation in particular [17]. It also demands that the model is developed in an environment that gives the modeller as much support as possible in the process of correlating observables and identifying and specifying the relationships between them.

The tkeden interpreter, developed by Simon Yung, is currently our most effective tool for addressing these issues. The features that tkeden supplies for the development of definitive scripts and associated agent actions do not lend themselves easily to the explicit representation of higher-order observables, but greatly simplify the task of identifying such observables. The text of definitive scripts is maintained within the tkeden environment in such a way that different categories of definition can be separately displayed, references to variables can be automatically located, dependencies can be more

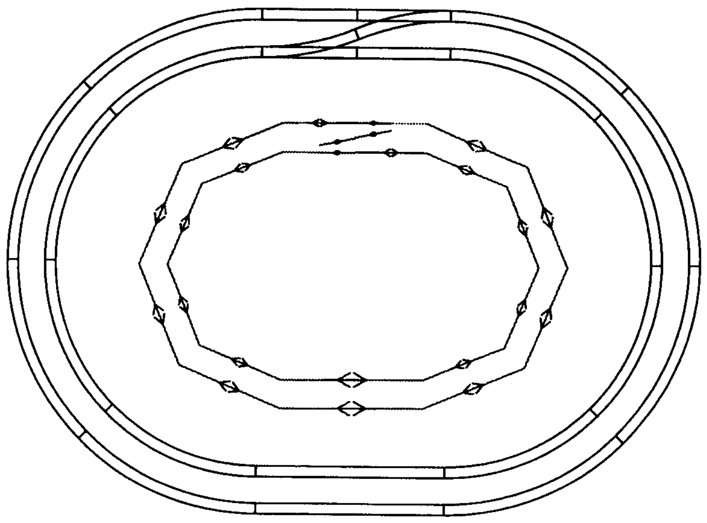


Figure 5: Different Designers' Perspectives on a Railway Track Layout

readily examined, undefined variables can be isolated and updated definitions can be recorded.

We believe that our Empirical Modelling approach is in principle particularly well-suited to dealing with concurrent engineering issues. Rationalising the interaction between independent agents in a concurrent engineering context involves two complementary concerns: maintaining coherent models of the partially dependent viewpoints of different design participants, and supplying a framework of interfaces and protocols for communication in the design process. Such rationalisation is needed to legitimise the concept (introduced in [1]) that the concurrent engineering process revolves around the cooperative development of a virtual prototype. Higher-order definitions can have a significant role in expressing the commitments associated with convergence between viewpoints, but it is evident that - in general - the kind of relationships between models that arise in this connection can only be satisfactorily expressed with reference to agents. By way of illustration, the relationship between the geometry and the connectivity of a model railway track depicted in Figure 5 is too subtle to allow a simple method of synchronising incremental changes to both models.

The tkeden interpreter offers some support for the design process through intelligent script management, keeping histories of the interaction, maintaining different versions and making a distinction between forms of definition that have different status. For example, because all interaction is construed as definition within the definitive paradigm, both editing a script and clicking the mouse can generate a definition. In order to reconstruct the current state of a design we do not generally need to record where mouse clicks have occurred, but would want to record more significant interaction. Such considerations motivate a hierarchical agent structure that admits different levels of definition and interaction. To put this in a concurrent engineering context, we shall need to describe agency that embraces interaction at all levels in a hierarchy ranging from low-level interaction through a conventional user-interface, through script editing, to version management, to communication between design participants, to management and monitoring of design team interaction. For this purpose, we shall require more sophisticated variants of our agent specification language LSD that can account for manager agents and the corporate action of agents.

7 Implementation Prospects

The implementation of a definitive programming framework that can support our requirement is exceedingly challenging. At a high level of abstraction, the distinction between values of observables, dependency structures and agent privileges is very conspicuous, yet consideration of simple scenarios for simulation indicate that these can all be interrelated in conceptually indivisible changes of state. For instance, when the liquid in which two objects are suspended becomes frozen, this can simultaneously change the values of observables, introduce a structural dependency and affect privileges for agent interaction. Modelling synchronised change in such diverse aspects of a phenomenon is difficult within our present framework. There are also severe computational overheads that limit the usefulness of our methods for advanced applications, such as geometric modelling [20].

One possible implementation strategy currently under investigation is the application of definitive principles at the lowest possible level of abstraction in a conventional machine architecture. This is the concept behind the DAM (definitive assembler) machine, currently being implemented on the ACORN RiscPC platform. In the DAM machine, dependency relationships can be established between the words in a definitive store. A DAM word can represent a value, an address or an instruction, and these are the primitive elements from which observables, structures and privileges are respectively constructed. On this basis, there are prospects for establishing indivisible associations between heterogeneous ingredients of a model. The DAM architecture also opens up the possibility of more direct utilisation of raw processing power in the maintenance of definitive scripts and implementation of agents. This applies both to the interpretation of definitive scripts, which can be directly refined into dependencies between structured collections of words in the DAM machine, and to the visualisation of definitive variables, where we hope to specify the relationships between visual elements of the screen definitively in a manner analogous to that used at a higher-level of abstraction in Simon Yung's SCOUT notation.

Our current research into geometric modelling aims to exploit the R-function representations of Adzhiev et al. [21] in conjunction with definitive scripts over the DAM machine to establish a direct correspondence between the visual display and the presence of material at positions in Euclidean space. The definitive script, specified using higher-order definitions, represents a pair of filters that together specify a visualisation of a 3D solid object. The first of these filters uses the implicit function representation of a solid object to determine the visible points on its boundary. The other attaches a definition to each pixel in an area of the screen display to represent brightness levels at a point projected from an eye position through the viewing plane in which the pixel resides. In this definition, the brightness levels are specified as a cross product of a light vector and a tangent to the surface of the 3D solid. With a careful use of high order definitions, only those pixels which are affected by a change in the defining parameters of the object will be redrawn. As all defining parameters and visualistion parameters exist within the definitive store, each can be redefined to fine tune the visualisation instrument into one which reflects as closely as possible a particular user's insight into the model.

Such use of definitive scripts in implementation potentially provides a powerful environment for computer-based visualisation. Experimentation with the parameters in such a script can be used to develop better models of the application, to improve the metaphor for visualisation and to optimise the use of the hardware and graphics platform as a visualisation instrument. The way in which the mathematical model of a geometric object is used in this visualisation process is unusual. By way of illustration, in our conventional implementation of geometric entities, a line (as in DoNaLD and CADNO) is represented by a pair of distinct points and some combinatorial information to record their interconnection. In our implementation of a line over the DAM architecture, the mathematical model serves a different role - implicitly establishing a direct link between the visual image of a "real-world" line and a pixellated line on the screen display. A paradoxical quality of our pure definitive implementation is that it mediates between lines as perceived in the real world and lines as groups of illuminated pixels on the screen. In this way no explicit reference is made to lines as mathematical abstractions. In particular, both real-world lines and lines on the screen have thickness and colour.

The concept of a higher-order definition also impacts at both ends of the modeller-machine spectrum in an intriguing way. The use of R-functions as a geometric representation is itself a form of higher-order definition that differs radically from the use of DoNaLD or CADNO. In this context, the correspondence between the components of an R-function expression and the features of the geometric object it represents is oblique, and lends itself to exploratory modelling that puts the emphasis upon the computer as a geometric instrument [21]. At the implementation level, a complementary process operates, whereby the values at the lowest levels of abstraction are not typically those that have a direct visualisation, but instead supply the internal machine representations that inform the display.

8 Conclusion

The combined use of definitive scripts and agents within an Empirical Modelling framework is a technique with considerable promise and power. In comparison with conventional modelling environments, tkeden offers the modeller opportunities for interaction of exceptional subtlety and flexibility. Interaction of this nature is particularly valuable in tasks such as conceptual design or requirements capture that demand exploration and experiment. To exploit these principles fully in advanced environments for geometric modelling and concurrent engineering, it will be necessary to provide means for the user to constrain agent interaction with the model through imposing structures upon systems of observables and protocols. The ongoing research outlined in this paper suggests methods by which this can be achieved. Our present objective is to rationalise these methods within a unifying theory of reference that will in due course inform yet more sophisticated versions of our current modelling tools.

9 Acknowledgements

We are indebted to the EPSRC for support under grant GR/J13458, and for its research studentship awards to Richard Cartwright and Dominic Gehring. We are also grateful to Matra Datavision for sponsorship via the CASE award scheme.

References

[1] V. D. Adzhiev, W. M. Beynon, A. J. Cartwright, Y. P. Yung, *A computational model for multiagent interaction in concurrent engineering*, Proc CEEDA'94, Bournemouth Univ., 1994, 227-232

[2] W. M. Beynon *ARCA: a notation for displaying and manipulating combinatorial diagrams*, University of Warwick Computer Science Research Report #78, July 1986

[3] W. M. Beynon, D. Angier, T. Bissell, S. Hunt, *DoNaLD: a line drawing system based on definitive principles*, University of Warwick Computer Science Research Report #86, October 1986

[4] W. M. Beynon *Definitive Principles for Interactive Graphics*, NATO ASI Series F, Vol 40, Springer-Verlag 1988, 1083-1097

[5] W. M. Beynon, M. D. Slade, Y. W. Yung, *Parallel computation in definitive models*, in Proc Conpar'88, British Computer Society Workshop Series CUP 1989, 359-367

[6] W. M. Beynon, M. T. Norris, R. A. Orr, M. D. Slade, *Definitive specification of concurrent systems*, Proc UKIT'90, IEE Conference Publication 316, 1990, 52-57

[7] W. M. Beynon, I. Bridge, Y. P. Yung, *Agent-oriented modelling for a vehicle cruise controller*, Proc ESDA Conf., ASME PD-Vol. 47-4, 1992, 643-8

[8] W. M. Beynon, A. J. Cartwright, *Agent-oriented modelling for engineering design*, Proc CAD'93, New Information Technologies in Science, Engineering and Business, Yalta, May 1993, 49-53

[9] W. M. Beynon, R. I. Cartwright, *Empirical modelling for principles for cognitive artefacts*, from Proc of IEE Colloquium on Design Systems with Users in Mind - the Role of Cognitive Artefacts, Dec. 1995

[10] A. Borning, *The programming language aspects of ThingLab, a constraint oriented simulation laboratory*, ACM Transactions on Programming Languages 3(4), 1981, 353-391

[11] A. J. Cartwright *Applications of Definitive Scripts to Computer Aided Conceptual Design*, PhD Thesis, University of Warwick, 1994

[12] M. Chmilar, B. Wyvill, *A software architecture for integrated modelling and animation*, New Advances in Computer Graphics: Proc of CGI'89, 257-276

[13] S. van Denneheuvel, *Constraint-solving on database systems: design and implementation of the rule based language RL/1*, CWI Amsterdam, 1991

[14] D. K. Gehring, *Tables as Definitive Variables*, University of Warwick Computer Science 3rd Year Project Report, May 1995

[15] M. L. Ginsberg, D. E. Smith, *Reasoning about action I: a possible worlds approach*, Artificial Intelligence 35, 1988, 165-195

[16] M. L. Ginsberg, D. E. Smith, *Reasoning about action II: the qualification problem*, Artificial Intelligence 35, 1988, 311-342

[17] D. Harel, *On visual formalisms*, CACM, 31(5) 1988, 514-530

[18] M. Heller, *Advanced Windows Programming*, New York, J. Wiley, 1992

[19] R. Popplestone, T. Smithers et el, *Engineering design support systems*, IKBS/MS 7, 1986

[20] A. Pasko, V. Adzhiev, A. Sourin, V. Savchenko, *Function representation in geometric modelling: concepts, implementation and application*, The Visual Computer 11:429-446, Springer-Verlag 1995

[21] M. J. Wooldridge, N. R. Jennings, *Intelligent agents: theory and practice*, Knowledge Engineering Review, vol.10, no. 2, 1995

[22] B. Wyvill, *An interactive graphics language*, PhD Thesis, Bradford University, 1975