A New Computer-Based Tool for Conceptual Design

Valery Adzhiev*, Meurig Beynon*, Alan Cartwright†, Yun Pui Yung*

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

ABSTRACT

We regard conceptual design as a negotiation between *what we believe* and *what we observe*. This process is not necessarily goal-directed, but establishes new connections between observables that become useful in product design. We propose an agent-oriented modelling method that can represent observation and experiment more effectively than conventional computational models. Our computer model is a metaphor for a physical prototype with which many design agents can interact from different perspectives. In this way we relate conceptual design to the coordination of experimental activity in the concurrent engineering process.

Our models make use of special-purpose *definitive scripts*, which permit synchronised update of variables: a generalisation of the spreadsheet principle. The definitive scripts are implemented in the environment of the Abstract Definitive Machine which provides an operational interpretation of the scripts. The use of these tools is illustrated with reference to the design of a model railway layout.

Introduction

Computer-based tools to assist the design of a genuinely new product are rare. Most are only useful downstream of the initial concept, accessing and manipulating existing knowledge in various ways. We are developing a tool that aids the designer in formulating hypotheses on the basis of information gained from experiment – a process that is characteristic of conceptual design.

Our approach is of particular interest in connection with concurrent engineering, and can be used to enable co-operating designers to construct a computational model whose role in the design process resembles that of a physical prototype. Within our computational model, we represent the characteristic mode of interacting with a prototype associated with each designer. In this way, each can simulate the effect of experimentation on a prototype, discovering and exploiting its relevant properties. We can also manage interaction with the model so that

several designers can be active simultaneously, and that the combined effect of their actions is visible at all times.

Our modelling method exploits a new form of agent-oriented modelling based on representing agent actions by redefinition of variables in the context of a script of definitions. This principle is, in essence, similar to the synchronised update of cells in a spreadsheet, but has been developed in our research so as to be completely general in its application. In particular, we can deal with every aspect of the design task by taking different interpretations for agents. For instance:

- in co-ordinating the design, the agents are the parties in the concurrent engineering process;
- in specifying the interface to a particular party, a typical agent is a standard procedure invoked through the selection of a menu button;
- in simulating use of the engineering device being designed, an appropriate agent is a component of the device.

Important features of our approach include:

- principled modelling that is intimately connected with observation and experiment in the physical world;
- effective ways to represent synchronisation and concurrent action;
- potential for integration with existing CAD methods and tools.

A synergistic advantage is that the construction of our computational model gives considerable insight into the patterns that underlie the design problem, offering the designer a structured way through the conceptual design maze by highlighting layers of abstraction behind each particular design realisation. We shall illustrate these features with reference to a case study based on model railway layout design.

What is conceptual design?

Conceptual design is an activity of broad and elusive scope. The Pocket Oxford Dictionary definition of *conceptual* – "of mental conceptions" – indicates that conceptual design is concerned with a process of abstracting from experience. Such a process is hard to formalise because it contains a cognitive heuristic component [SN89]. In this paper, we regard conceptual design as the negotiation between *what we believe to be true* and *what we observe to be true* that is involved in developing concepts consistent with experience. Activity of this nature operates at many different levels of abstraction and in many aspects of the design process.

Our principal aim is to illustrate how new principles for computer-based modelling can assist conceptual design. Our illustrative examples are necessarily concerned with the solution of specific design tasks, but the modelling principles we consider are rooted in an experimental method that is not narrowly goal-directed. This accords well with our view of conceptual design as a non-routine, creative process in which interaction is essential, and unexpected invention may result.

Observation and Experiment in Conceptual Design

Conceptual design activity is essentially concerned with the development of mental conception through experiment with physical prototypes[1]. The fact that in practice a designer may perform few actual experiments with physical objects should not disguise this underlying principle. At some level of abstraction, the evolution of mental conceptions and the construction of physical prototypes interact in the conceptual design process. And though both may be represented in an abstract manner (e.g. through computer-based modelling), they are activities of a fundamentally different character.

In constructing physical prototypes and experimenting upon them, the designer is engaging with experience, and is fundamentally concerned with how the world *appears to* behave. In developing mental concepts to organise this experience, the designer is framing a hypothesis about how the world *can, will and must* behave. For the truly open-minded, there are no such hypotheses – only the evidence of past experience to guide and of future demonstration to confirm.

In our approach to design, we introduce computer-based modelling methods that can be used to simulate the process of constructing physical prototypes and performing experiments upon them. The development of these models relies upon a view of computation that is fundamentally different from orthodox computer programming and formal specification. The distinction between our modelling methods and traditional approaches to computer programming reflects the profound difference between theorising and experimenting.

---

[1] The word *prototype* is used here to refer to a physical model that imitates some of the characteristics of the design object, and can be used to investigate it in this respect. For example, a clay model of a car is a prototype that can be used to assess its appearance.

Computational models

Computer models to represent generalisations from experience

The mental conceptions developed in conceptual design are generalisations from experience. As examples, we can cite objects, patterns of behaviour and constraints that the designer identifies in the design process. An object-like abstraction is appropriate where the same kind of possible configurations and transformations apply to many different families of experimental observations. In creating an object abstraction, we express faith in the persistence and integrity of a collection of observables and the range of possible states and transformations that is of relevance. In the design context, the basis for this faith typically has to be negotiated by limiting the range of possible experimental environments and interactions considered. For instance, in the design of an aeroplane, we do not consider the possibility that painting the wings a different colour will materially influence aerodynamic performance, nor explore the likely consequences of transforming the wings to a molten state.

Conventional computational models rely upon abstractions for capturing generalisations from experience. For instance, objects, patterns of behaviour and constraints are the key concepts in object-oriented programming, in formal specification and in logic programming respectively. These abstractions have been developed to represent well-specified and precisely circumscribed behaviours of reliable computing devices. The fact that they are ill-suited for representing experience from the experimenter's perspective limits the usefulness of conventional programming methods in conceptual design (cf. [Tomiyama89b, Koegal89].

Computer models to represent observation and experiment

The fundamental objective of our model-building is to exploit the power of the computer to imitate the perceived states and primitive state-transitions of the external world. By *perceived state*, we here refer to a particular set of measurements of observables such as might be made in a single experimental reading. By a *primitive state-transition*, we refer to the changes to these measurements that would result from a single "atomic" action on the part of the, such as changing the value of a control parameter. The key principle behind our modelling is that there should be a direct correspondence between values in the computer model and observables in the external world, and that this correspondence should reflect the way in which changes to the values of observables are synchronised when the state is changed.

The computational tools we have developed are designed with the direct representation of experimental observations in mind. The fundamental idea behind our approach is that of representing observables by variables, and introducing **definitions** (similar in character to the

defining formulae for cells of a spreadsheet) to express the way in which the values of observables are interdependent. Such a set of definitions – a **definitive script** – represents a possible experimental observation, and *redefining a variable* corresponds to *changing an experimental parameter*. Our methods are discussed in detail elsewhere [ABCY94, BBY92, BSY88, BY88, BY92, FBY93, NBY94], and will be illustrated more fully below.

The faithful correspondence between the values of cells in a spreadsheet and the external physical quantities they typically represent is a familiar illustration of the modelling principles we have described. A more direct way to establish such a correspondence between the state of a computer model and the state of an external system is to use a generalised form of the spreadsheet principle to construct a graphical image in which the dependencies between geometrical elements are specified by a definitive script. Illustrative examples drawn from our previous case-studies include scripts to represent the floor plan of a room [BY88], the disposition of a sailboat [NBY94], or the dial of a speedometer [BBY92]. An image specified in this way is not merely a pictorial representation of an external object, but uses a geometric metaphor to imitate a mode of observation of the object. Such a representation establishes a link between form and content similar to that found in the spreadsheet, where the relationships between the values of cells serve to identify it as a model of a particular external set of observables in a manner that can be tested objectively.

Contrast between the computational models

A definitive script as a model of an experimental observation represents a primitive hypothesis about how the world behaves. To convince another person of the validity of my model, it suffices to establish that we see the same observables, can agree about their current values and share the same expectations about the immediate effect of changing parameters. Note that, unless and until the set of redefinitions that can be applied to my script is precisely specified, there is no absolute commitment on my part to a particular model of my observations. This is in keeping with the fact that the expectations developed through experiment are always subject to falsification, and are asserted subject to faith in prediction from past evidence. Expectations can be confirmed and confounded – but never justified – by experiment.

The qualities of a definitive script are quite different from those of a logical specification of properties. Logical statements are interpreted in isolation from any specific context, and are totally abstracted from content. A definitive script is to be interpreted in relation to an external situation, and has no significance without reference to the experimental context it represents. Logical statements express constraints about behaviour that are based on the presumption of comprehensive knowledge of what might be the case. Definitive scripts express tentative hypotheses about primitive behaviour in tightly constrained circumstances. Logical statements

are intended to express objective knowledge that is not context-specific, whereas definitive scripts are formulated with a particular view of data in mind. Logical statements are appropriate where we have sufficient experience and conviction about behaviour to be able to infer the outcome of a "what if?" experiment. Definitive scripts are appropriate when the relationship between observables is still subject to exploration.

Behavioural specifications

A typical engineering design problem involves the construction of an object with a rich autonomous behaviour. By itself, a definitive script describes a state in which the user is able to act through redefinition to change the state as an might. In order to extend our computational models to permit the specification of behaviour, we must introduce other state-changing agents into the model. This can be done most effectively by exploiting the same principle of representing a state by a definitive script and expressing each state-change as a redefinition.

The consideration of other agents draws attention to the subjective nature of the dependencies between observables. In some contexts, it can be argued that agents are already implicit in a definitive script, and that their actions are responsible for the propagation of state-changes between observables. For instance, in pulling a lever that propagates through a mechanism to perform action at a distance, there will be some delay in the response due to the tolerances in the linkages. Such a delay might be conceptually insignificant in relation to the actions of a human user, and most appropriately represented by a definitive script. In contrast, it would be very significant where synchronisation with a electrical signal propagated in parallel was concerned, and would be more appropriately described in terms of interactions between agents.

The need to consider the behaviour of a system from the perspective of many different agents leads us to specify the role that each agent can play in interacting with observables through stimulus and response. A special-purpose notation LSD is used to describe this interaction with reference to

- what observables each agent receives as a stimulus,
- what observables each agent conditionally has the power to change,
- by what protocol each agent is privileged to initiate changes.

Full details of the LSD notation are described elsewhere [BBY92, BY92]. Further references to its use in specifying behaviours and in describing the interaction of design agents in the concurrent engineering process are made below.

Tools for conceptual design

The principal thesis of this paper is that conceptual design is closely connected with the process of developing theories following the traditional scientific method. In certain aspects of conceptual design, as when probing a novel direction for development, the designer has to carry out work of an exploratory nature, with no explicit goal beyond the faithful modelling of phenomena. The first objective of such activity is to find consistent patterns, irrespective of utility or purposeful exploitation of these. Such experimentation may perhaps be deemed design only where there is a design artefact, but plays a part in every design process.

The software tools we have developed are best understood with reference to the ingredients of the experimental process. Because of the breadth of concern in engineering design (e.g. the need to take account of environmental effects, human factors and non-functional requirements), we need a broad notion of *experiment*. When describing an experimental context, we consider

- the choice and the nature of the observables,
- the characteristics of the autonomous agents that are present,
- the scenarios for agent action that are presumed to be operative.

Each of these ingredients has its counterpart in our computer model of an experimental context. The choice and the nature of the observables is captured by the variables present in a definitive script. The characteristics of the autonomous agents that are present are expressed in the LSD specification. The scenarios for agent action that are presumed to be operative are invoked when simulating the behaviour of a system of interacting agents. This simulation is carried out in the computational model of the Abstract Definitive Machine (ADM), in which the state of the system is recorded in a dynamically changing script of definitions. Full details of the ADM are given elsewhere, but further illustrations of its use are informally described below [BSY8].

The experimental context for conceptual design

We have referred to conceptual design as a process of negotiation that is closely associated with the identification of an experimental context. We must decide what is to be observed, what agents are presumed to act, and what environmental constraints govern their interaction. By way of concrete illustration, the concept of an aeroplane can be understood with reference to the way in which we view various aspects of its behaviour.

Consider for instance the experimental observations that might be made of a aeroplane to determine:

1. its fuel consumption on take-off into a 50 mph headwind,
2. its behaviour on crash landing in the sea,

3. how its wings reflect the light at sunset,
4. how it reacts to being immersed in liquid nitrogen.

These aspects of aeroplane behaviour belong to quite different categories:

1. routinely happens and is carefully analysed and observed,
2. happens very infrequently, is not a part of normal routine function, but is relevant to the design as a possible exceptional behaviour,
3. routinely happens but is not systematically observed and doesn't influence the design,
4. is quite inconceivable, and lies far beyond the scope of concern.

The purpose of this classification is to illustrate how the concept of an aeroplane is defined with reference to a set of observations and environments that are relevant to its characteristic function. Conceptual design is concerned with identifying and exploring relevant observations prior to formulating the hypotheses upon which the design of an aeroplane rests. By its nature, this process of identification demands that we explore aspects of behaviour that are ultimately deemed to be outside the design space.

In identifying the concept of aeroplane, we are concerned with the question: *Who is the observer?* We don't design planes for poets who watch them at sunset, or for malicious agents from Mars who drop them into liquid nitrogen. The actual observable behaviours of an aeroplane can't be circumscribed though, and to some extent new insights in design are likely to be associated with taking account of behaviour not previously considered. For instance, the Wright brothers didn't consider how noisy their plane was, or how to their plane and pilot would respond to turning over in flight.

Our discussion shows how convergence on an object concept is a matter of *agreed interpretation.* To establish such a concept, we need a consensus about how the object is to be observed, how it's then observed to behave, and what part of the behaviour is characteristic of the object. This leads us to say *what an aeroplane is* in terms of how we observe it, and where we can conceive it fulfilling its function. In effect, we circumscribe that part of the behaviours 1,2,3,4 we are concerned with: we are not interested in appearances at sunset or the effect of immersion in liquid nitrogen etc.

The computational context for conceptual design

The basis for the experimental method lies in the correlation between observations and computations made in an independent frame of reference. It is because we can sometimes compute the value of an observation from knowledge of the values of others that we get useful information from an experiment. In our modelling framework, the role of the computer is to supply the independent state-changing system with which the results of experimental readings

can be correlated. The way in which this is done depends upon what metaphors our computer device makes available to us.

The metaphors we can exploit are determined by the kind of computer states that can be specified using definitive scripts. For instance, in a spreadsheet, the scalar values of cells are the metaphor for observations. The definitive notations that we have developed for graphics, text and window layout enable us to specify the state of the computer screen in a manner that directly imitates external observation. In principle, definitive notations to permit the construction of computer-generated multi-media models would be valuable.

In an experimental context where autonomous agents are active, it is also necessary to provide interfaces through which to simulate their views and actions. These mechanisms must also rely on metaphors that reflect each agent's perspective as faithfully as possible. The paucity of mechanisms typically available for this purpose is evident when we consider the difficulty of modelling the adjustment of the sheet on a sailboat [NBY94], or the motion of a billiard cue [FBY93].

We use the term *construction* to describe the computer model of an experimental context that is generated by our method [NBY94]. A construction typically contains a definitive script to represent the relevant observables, LSD specifications to represent the agents that are active in the environment, and one or more interfaces through which relevant behaviour can be simulated. Our previous work gives many examples of such constructions [BBY92, BY88, FBY93, NBY94]. What is expressed by a construction has an ambiguous quality; it can be interpreted as the direct representation of experimental observation or as representing expectations about the observed experimental behaviour. Depending upon which perspective we adopt, a construction may be regarded as open to further development through refinement of the model or as a fixed specification of the design requirement.

Constructions can be viewed as implementing frames, in the sense of Minsky [Minsky88]. In particular, the development of constructions is directed by expectation, elaboration, alteration, novelty and learning such as is discussed in [Minsky88]. The merits of using constructions as basic computational building blocks emerge when we consider how readily definitive scripts can be composed to reflect observations made in two or more experimental contexts. Our experience suggests that the use of scripts helps to address the well-recognised problem of unifying different representations in design (see [Tomiyama89a]). Where conflicts arise, it is because of the interaction between two different experimental perspectives – a central issue for conceptual design.

Goal-directed experimentation in design

We now consider the role played by the processes of experimentation we have described when the aim is to produce a particular design artefact using our computational tools. Our approach may be compared with that described in [Veerkamp92], where the design task is also hierarchically structured, the role of constructions is played by the *fact* and *object bases* and the experimental component is represented in *scenarios*.

In our approach, the design process is represented via the development of many different frameworks for observation of the design artefact. Each such framework is represented by a construction, and is associated with a particular perspective on the design. In a concurrent engineering context, these perspectives may belong to different participants, but they will typically be present even where there is a single designer, as they represent different valid – and quite possibly conflicting – perceptions of the design goals. In either context, we distinguish different design perspectives by attributing them to agents, and regard the design process as involving the co-ordination of their design activities. To give a proper account of the design process, it is appropriate to discuss the design process with reference to many participating *design agents* whether or not these roles are performed by different designers.

Different ways in which to enhance and transform the design artefact are associated with each perspective. These are represented by the privileges to redefine variables in the script that are associated with the various design agents. Where appropriate, the behaviour of the design artefact is itself modelled in terms of patterns of redefinition associated with autonomous agents. In our design of a Vehicle Cruise Controller, for instance [BBY92], we introduce agents whose behaviour is event-driven to represent the components of the engineering device.

The products of the design process are scripts that together embody a *virtual prototype* for the design artefact. The formal status of the virtual prototype can only be appreciated with reference to an official or consensus view about the current state of development of the design artefact. Each design agent will typically develop many definitive scripts, and perform many private experiments concerned with the selection and optimisation of particular features. A conceptual framework is needed to express the manner in which the contributions of particular agents are integrated in the final design. A full account of such a framework is beyond the scope of this paper, but its principal components will be briefly summarised.

It is convenient to describe the form of the concurrent engineering process as it would operate when the management of the design process for a design artefact was well-understood. In practice, when a novel design artefact is first created, the design process has itself to be discovered, and the patterns of interaction that lead to its development cannot be preconceived.

(The design of the design process is itself a subject for experimental development similar in nature to that we have discussed above. Our agent-oriented approach is sufficiently general to allow us to represent this meta-design activity, cf. [ABCY94].)

Two fundamental problems must be addressed in the concurrent design process: the synchronisation of changes in the evolution of the prototype, and the distribution of responsibility for managing its systematic development. The objective of the design process is the synthesis of a prototype by the design agents over time. The prototype is represented by a set of *constructions* (cf. [NBY94]) that become more coherent as the design progresses. Amongst these constructions we single out one in particular that provides a specification to be met by the final artefact. This specification includes both preconceived and evolving design constraints.

Because of the need to resolve conflict in the concurrent design process, the roles of the design agents have to be organised in a hierarchical manner. The co-operative activity of a set of design agents involves the co-ordination of their independent activity in developing constructions that reflect different experimental contexts for exploring the design artefact. The design process is then organised around a hierarchical "top-down" decomposition of the principal design task, complemented by "bottom-up" synthesis of the prototype.

Developing a consensus amongst the design agents about the nature of the design artefact is an essential part of the process of synthesising the virtual prototype. In arriving at a consensus view, features of the design artefact change their status in the same way that experimental observations mutate into theoretical predictions. This aspect of the design process has to be described with reference to time, and to stages that lead to milestones after which immutable assumptions about the design artefact have been adopted.

In our agent-oriented framework, we can best describe the evolution of the design as performed by *design co-ordinators* operating at different levels of abstraction in the task hierarchy [ABCY94]. The role of a design co-ordinator bridges the gap between *what must be true of the design artefact* – in some aspect – and *what could possibly be true* according to the proposals of design agents at the next level of abstraction in the task hierarchy. In explicit terms, the function of the design co-ordinator is to amalgamate constructions through arbitrating where there is conflict – a process that typically involves fixing common parameters derived from independent experimental contexts. The design co-ordinator then serves as a design agent who supplies a construction to a co-ordinator at a yet higher level of abstraction.

The role of a design co-ordinator is to identify the stages and milestones, to declare the patterns of interaction amongst the design agents, and to further the design process beyond each

milestone. Such activity is central to the theme of this paper, since it involves the generation of a mental conception of the design artefact through the co-ordination of experimental activity at a lower level of abstraction.

A Case Study: model railway layout design

As an illustrative example, we consider the design of a model railway layout. In one context, the design may involve the use of a standard catalogue of track segments; in another, the construction of a track from first principles. When designing a track layout from standard components without the aid of a computer, the designer will typically consult previous track designs for preliminary ideas, but will also assemble components in experimental configurations to assess their feasibility and suitability. Small-scale replicas of standard track segments are available for this purpose, and can be used to construct a prototype layout. The process of experimenting on the prototype is significant in that

- more information can often be extracted from the physical model than was anticipated when the model was first conceived,
- it is essentially interactive in nature, and cannot be fully automated because it relies upon feedback to the designer at regular stages.

In our computer-based modelling approach, we can develop a graphical display to simulate the process of prototyping a layout. Figure 1 depicts a layout that is specified using a "definitive script" in which each variable corresponds to some attribute of a track segment. Track segments are assembled by introducing definitions to locate the start of one segment at the end of another. In our model, track segments are represented by their start and end reference points and their characteristic identification number, from which physical attributes (such as curvature and length) can be inferred.

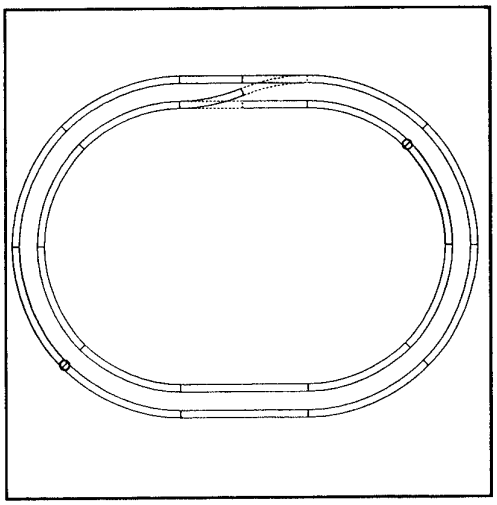

Figure 1: A graphical representation of a track circuit

An extract from the script that defines Figure 1 is given in Listing 1. Each individual track segment is derived by geometric transformations from a standard prototypical part. In this context, the design process is based on the selection and assembly of parts from a standard catalogue, but admits a non-routine aspect associated with experimental layout design (cf. [PKM92]).

An alternative approach to track layout design might be based on the construction of track segments themselves. Such an approach gives far greater flexibility to the designer, making it possible to use segments with arbitrary length and curvature. By introducing scripts that represent the individual track segments in Figure 1 explicitly, as illustrated in Listing 2, we arrive at an alternative representation for Figure 1. Note that in this script the parameters for each track segment are specified independently, and can also be redefined independently. Redefinitions of this nature offer greater potential for transformation of the layout, associated with more ambitious experiments.

```
# standard 45° 2nd radius track
openshape ST226
within ST226 {
  real angle
  angle = pi div 4
  line f, e
  f = [{~/TrackWidth div 2 @ inDir - pi div 2},
     {~/TrackWidth div 2 @ inDir + pi div 2}]
  e = rot(f, start +
      {~/Radius2 @ inDir + pi div 2}, angle)
  arc l, r
  l = [f.1, e.1, -angle * 180 div pi]
  r = [f.2, e.2, -angle * 180 div pi]
}

# the first circular piece to the left of the set of
# point in the outer ring
openshape A9
within A9 {
  point start, end
  real inDir, outDir
  start = ~/A10/end
  inDir = ~/A10/outDir
  shape track
  track = trans(rot(~/ST226, {0,0}, inDir),
        start.1, start.2)
  end = rot(start, start + {~/Radius2 @ inDir
        + pi div 2},~/ST226/angle)
  outDir = inDir + ~/ST226/angle
}
```

Listing 1: *Definition of a track segment based on transformation from a standard prototypical part. An extract from the script producing Figure 1.*

```
# the first circular piece to the left of the set of
# point  in the outer ring
openshape A9
within A9 {
  point start, end
  real inDir, outDir
  start = ~/A10/end
  inDir = ~/A10/outDir
  real angle
  angle = pi div 4
  line f, e
  f = [start + {~/TrackWidth div 2 @ inDir
      - pi div 2}, start + {~/TrackWidth
      div 2 @ inDir + pi div 2}]
  e = rot(f, start + {~/Radius2 @ inDir
      + pi div 2}, angle)
  arc l, r
  l = [f.1, e.1, -angle * 180 div pi]
  r = [f.2, e.2, -angle * 180 div pi]
  end = rot(start, start + {~/Radius2 @ inDir
        + pi div 2}, angle)
  outDir = inDir + angle
}
```

Listing 2: *An alternative, explicit representation of the track segment A9. The variables angle, f, e, l and r are now local properties of A9. In contrast to Listing 1, redefinition to these variables will not affect the shape of other track pieces.*

The use of definitive scripts and redefinitions to simulate experiments on physical prototypes is a general theme in our research [ABCY94]. The variables in the script correspond to observables, such as the length and curvature of the track segments in the layout depicted in Figure 1. Of course, the physical layout has many attributes not represented in Figure 1: such as the weight, colour or cost of each segment. Part of the design process is concerned with identifying what observables are significant for the design task, and this identification may itself involve discovery through experiment.

Within the experimental framework, it is useful for the designer to be able to change the mode of observation without completely rewriting the script. This is typically possible through extension of the script. As a simple example, to estimate the cost of constructing the layout in Figure 1, it is only necessary to introduce a price for each component, maintain an inventory of components as a list, and record the total cost of parts in the inventory. Notice that activity of this kind is normally outside the scope of conventional physical prototyping techniques, where different prototypes have to be constructed to represent different characteristics.

The script in Listing 1 has its limitations. Figure 1 reveals sets of track segments that make up a circuit, but this fact is reflected only in the visualisation, and is not explicit in the definitive script. The manner in which the layout is described in the script informally suggests a particular sequence of layout construction, whereby track segments are added one-by-one until a circuit has coincidentally been completed. Note that the presumption that a sequence of track segments create a circuit is based on the implicit assumption that the layout is 2-dimensional; in a 3-d model it would be reasonable for the track to spiral upwards to reach a new level. Considerations of this nature reveal the subtlety of the issues involved in replacing the acyclic geometric relationships between track segments in a definitive script by cyclic constraint relationships that might appear to be more appropriate representations for circuits.

In practice – to simulate the layout in use, for instance – it is essential for the designer to model the topology of the layout, as well as its physical appearance. To do this, the interconnection of track segments must be represented by more than physical coincidence of endpoints on the screen. A suitable representation for the topology of the layout in Figure 1 is displayed in Figure 2, in which a directed graph with coloured edges is used to depict the combinatorial relationships between track segments. In this digraph, the edges represent directions for the traversal of track segments. An extract from the definitive script that describes Figure 2 is given in Listing 3. In studying the relationship between train movement and the settings of points in the layout, the topological and geometrical models depicted in Figures 1 and 2 must be used in conjunction. For this purpose, Listings 1 and 3 are suitably integrated in a manner

that illustrates how a computer-based prototype can combine representations derived from several different modes of observation.

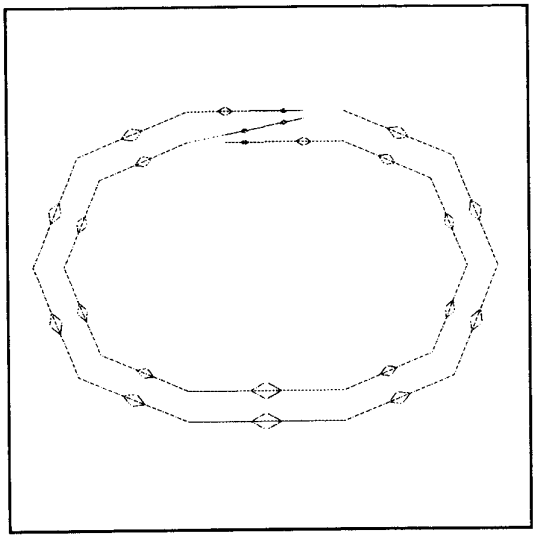

Figure 2: The combinatorial relationship between track segments

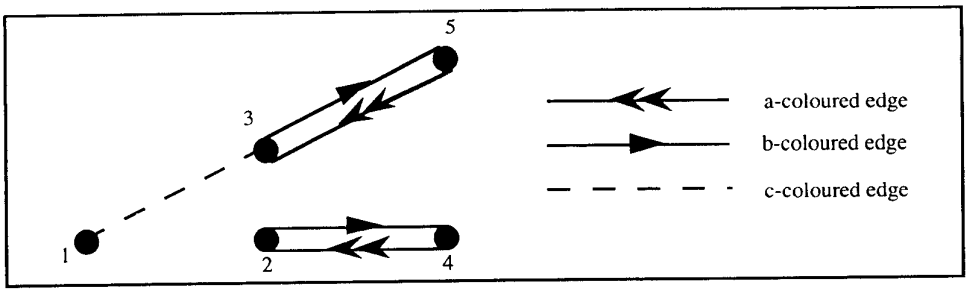

Figure 3: Combinatorial relationship within a set of points

```
mode Point1 = 'abc'-diag 5              # points have 5 vertices and 3 colours

Point1Stts = 1          # status of the points, 1 = closing the loop
# defining the edges
a_Point1{1} = 1;        b_Point1{1} = 1;        c_Point1{1} = if (Point1Stts == 1) 2 else 3
a_Point1{2} = 2;        b_Point1{2} = 4;        c_Point1{2} = if (Point1Stts == 1) 1 else 2
a_Point1{3} = 3;        b_Point1{3} = 5;        c_Point1{3} = if (Point1Stts == 1) 3 else 1
a_Point1{4} = 2;        b_Point1{4} = 4;        c_Point1{4} = 4
a_Point1{5} = 3;        b_Point1{5} = 5;        c_Point1{5} = 5

Scale = 100                                     # scaling factor for locating vertices
Point1!1 = A!1                                  # defining locations of the vertices of Point1 in
Point1!2 = A!1 - [Scale / 2, 0, 0]              # terms of the location of the first vertex of
Point1!3 = A!1 - [Scale / 2, Scale / 10, 0]     # A, the graph representing the outer loop
Point1!4 = A!1 - [Scale, 0, 0]
Point1!5 = A!1 - [Scale, Scale / 5, 0]
```

Listing 3: *An extract from the definitive script describing the combinatorial relationship between track segments.*

In the view of the layout shown in Figure 2, a set of points is represented by a graph consisting of 5 vertices and edges of 3 colours. An a-coloured edge can be interpreted in such a way that it always carries a train anti-clockwise in the track layout. Likewise a b-coloured edge carries a train clockwise. A c-coloured edge is an undirected edge that is conditionally connected to different inlets or outlets dependent on the status of the points. For instance, the definition

$$c\_Point1\{2\} = if\,(Point1Stts == 1)\ 1\ else\ 2$$

means that when the train is at vertex 2 in *Point1*, it can follow a c-coloured edge to vertex 1 if *Point1Stts* == *1*, or stops at vertex 2 otherwise.

As a final example of the way in which observation has to be interpreted with reference to additional domain knowledge, suppose that the layout is being designed for an electrical model railway. If the mode of operation of the model is such that return loops require special wiring, there is a need to identify whether such loops occur in the layout. A useful feature of modelling the state of the design explicitly is that the condition of "having a return loop" can in principle be automatically monitored, so that the designer is warned of the presence of such a loop, or even constrained to avoid introducing such a loop.

The simulation of model trains in motion introduces autonomous agents to the experimental environment. The operation of the layout involves two kinds of activity: continuous motion of trains in accordance with physical laws, and discrete events associated with starting and stopping trains and switching of points. In modelling continuous motion, we can introduce time as an extra observable, and describe the dynamics via a definitive script that is expressed in terms of *analogue variables* whose values vary as a function of time in ways that can be preconceived. Discrete-event modelling is described in terms of interactions between agents that are specified using LSD and simulated using the ADM.

In concurrent systems modelling using our agent-oriented approach, we explicitly specify the state changes by which information is communicated between cooperating agents. In this context, an important concept that relates to the theme of this paper is that of ritualistic action, whereby an action is invested with meaning by agreed convention. A particular example of this principle may be seen in [BY92], where the exchange of signals between guard and stationmaster associated with a train departure protocol is specified and simulated. Similar issues would be relevant to specifying protocols for transfer of control when several operators are in control of a layout.

The design of a model railway layout involves a wide range of issues: functionality, costing, wiring, aesthetics, usability, accessibility etc. It is clear that the interaction between different design perspectives can lead to many varieties of generalisation from experiment. Examples include:

- constraints on the layout design e.g. no return loops, a fixed layout size, restrictions upon gradients,
- constraints on how the design is carried out e.g. from free modelling of layout, we may move to an object-based paradigm with a catalogue of components, or formulate design rules and styles to ensure consistent overall layout characteristics.

These example illustrate how the character of the experimental contexts changes as design constraints are imposed.

The need to synchronise modelling of scenery with track design illustrates the essential role for co-ordination and for milestones in the layout design process. Several possible patterns of interaction in the design process might be used to resolve potential conflicts. The selection of the final design for some segment of track can be the result of building scenery to suit a specified track design, designing the track to follow a scenic model, or reaching a compromise imposed through co-ordination of both activities. Such issues demonstrate the significance of stages in the design process, and the manner in which the design process leads dynamically to the creation of experimental contexts.

Conclusions

The abstract ideas presented in this paper are based on a series of practical case-studies in modelling and simulation for engineering design [BBY92, BY92, FBY93, NBY94]. The spreadsheet-like principles we have exploited in our research have a long pedigree in graphics and design [Wyvill75, PSCKMS86, HM83], and have now come to play a prominent role in state-of-the-art application building tools (see e.g. [Nardi93]). We believe that the embryonic design framework we have identified can be built upon a new foundation for computation that is better-suited to the demands of modern computer systems engineering than the traditional mathematical theory of computation.

Experience with small case-studies has demonstrated that the incremental development of constructions is a powerful modelling technique that is relatively easy to learn and adapt. For instance, short assignments on an MSc module taught at Warwick in 1992-3 and 1993-4 together generated a dozen or so simulations of some practical subtlety and interest [NBY94, FBY93]. Our initial work on concurrent engineering has been promising [ABCY94], but we have yet to demonstrate the practicality of the design method described in this paper through a full-scale case study. To do this most effectively, we propose to extend the development of the Abstract Definitive Machine.

Acknowledgements

We are grateful to Paul Ness and Steve Russ for useful discussions. We are indebted to the SERC for support under grant number GR/J13458, and to the Royal Society for sponsorship of Valery Adzhiev under its Visiting Postdoctoral Research Fellowship Scheme.

References

[ABCY94] Adzhiev, V, Beynon, W M, Cartwright, A J, Yung Y P, *A Computational Model for Multi-Agent Interaction in Concurrent Engineering*, Proc. CEEDA'94, Brighton, 1994, to appear

[BBY92] Beynon, W M, Bridge, I, Yung, Y P, *Agent-Oriented Modelling for a Vehicle Cruise Control System*, Proc. ASME Conf ESDA '92, Istanbul 1992, 159-165

[BSY88] Beynon, W M, Slade, M D, Yung Y W, *Parallel Computation in Definitive Models*, CONPAR '88, British Computer Society Workshop Series CUP 1989, 359-367

[BY88] Beynon, W M, Yung, Y W, *Implementing a Definitive Notation for Interactive Graphics*, New Trends in Computer Graphics, Springer-Verlag 1988, 456-468

[BY92] Beynon, W M, Yung, Y P, *Agent-Oriented Modelling for Discrete-Event Systems*, Proc ICC Coll. "Discrete-Event Dynamic Systems", Digest #1992/138, June 1992

[FBY93] Farkas, M, Beynon, W M, Yung, Y P, *Agent-Oriented Modelling for a Billiards Simulation*, Research Report #260, Computer Science Department, University of Warwick, Dec 1993

[HM83] Hartquist, E E, Marisa, H A, *PADL-2 Users' Manual*, Cornell Programmable Automation, Ithaca, NY, 1983

[Koegal89] Koegal, J F, *Planning and Explaining with Interacting Expert Systems'*, Intelligent CAD Systems III, Eds V Akman, P J W ten Hagen, P J Veerkamp, Springer Verlag 1989, 17-31

[Minsky88] Minsky, M. The Society of Mind, Picador, London 1988

[Nardi93] Nardi, B A, A Small Matter of Programming, The MIT Press, 1993

[NBY94] Ness, P, Beynon, W M, Yung, Y P, *Applying Agent-Oriented Design to a Sail Boal Simulation*, Proc. ASME ESDA '94, London, to appear

[PKM92] Piela, P, Katzenberg, B, Mackelvey, *Integrating the User into Research on Engineering Design Systems*. Research in Engineering Design (1992) 3:211-221

[PSCKMS86] Popplestone, R J, Smithers T, Corney J, Koutsou A, Milington K, Sahar G. *Engineering Design Support Systems*. IKBS/MS 7/86 3.1

[SN89] Spillers, W R, Newsome, S, *Design Theory: A Model for Conceptual Design*, Proc. 1988 NSF Grantee Workshop on Design Theory and Methodology, Springer Verlag, 1989

[Tomiyama89a] Tomiyama, T, *Meta-model: A key to Intelligent CAD Systems*, Research in Engineering Design, Vol 1, No 1, 1989, 19-34

[Tomiyama89b] Tomiyama, T, *Object Oriented Programming Paradigm for Intelligent CAD Systems*, Intelligent CAD Systems III, Eds V Akman, P J W ten Hagen, P J Veerkamp, Springer Verlag 1989, 3-16

[Veerkamp92] P J Veerkamp, On the Development of an Artifact & Design Description Language, CWI Amsterdam, 1992

[Wyvill75] Wyvill, B, An Interactive Graphics Language, PhD Thesis, University of Bradford, 1975