Databases from an Agent-oriented Perspective

Meurig Beynon[*], Alan Cartwright[†], Yun Pui Yung[*]

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

abstract

We regard a database as a computer model that enables us to simulate different modes of interaction with a real-world system. We describe new agent-oriented modelling principles that can be used to construct such models. These principles use systems of definitions ("definitive scripts") to record the current values of real-world observables, and represent a far-reaching generalisation of the principles adopted in the relational language ISBL. An exercise in concurrent engineering: the design of a lathe shaft, is used for purposes of illustration. We compare our approach to data modelling with relational, deductive and object-oriented approaches.

1. Introduction

The relative merits of relational and object-oriented databases are currently the focus of great controversy. This paper examines database issues with reference to a new approach to data representation that has interesting connections with both relational and object-oriented models.

Our recent research at the University of Warwick has led to the development of new principles of agent-oriented modelling with applications to engineering design and software development [BC89, BY92, BBY92, BC92]. The mode of data representation we adopt has a central role. Wherever the state of an interaction between agents is to be described, it is captured using a system of interrelated definitions[1] of variables, or *definitive script*. A similar mode of data representation has been exploited by other authors in several systems for interactive graphics and design since its introduction by Wyvill [W75]. In the context of this paper, the most interesting feature of our work is that definitive scripts perform many of the functions of a conventional database, and provide generic techniques for representing the interaction between different agents and the data model.

[1]. Throughout this paper, the term "definition" is used in a technical sense, and "definitive" to mean "definition-based".

1.1. The Nature of Modern Databases

Modern applications of computers have radically changed our view of the function that a database can perform. An early textbook on databases defines a database as "a computer-based record-keeping system", but this definition is inadequate in at least two respects:

- the nature of the data stored in a modern database has changed – in response to the demands of multi-media, for instance. Even if we accept Date's view that – at some level of abstraction – "there is no known data that cannot be represented in tabular form" ([DD90], p6), it is clear that any general theory of data representation that treats the table as a fundamental primitive will have to invoke contrived data transformations.
- the manner in which data is presented to the user and manipulated by the user is an essential concern. For instance, a satisfactory theory of data representation should distinguish 'modifying parameters in an abstract table of numbers representing a complex geometric object' from 'rotating an image of the geometric object by direct manipulation'.

From a modern perspective, a more general abstraction for a database is appropriate: a database is a computer model that is a source of faithful views of the external world. At any stage, the state of the computer model corresponds to the actual or conceived state of an external system. The user perceives the state of the computer model via views that rely upon suitable metaphors. The table metaphor that works well for a traditional record-keeping system isn't a good way to describe a visual image. Modern computers have unexplored potential for new metaphors. Each metaphor involves different ways of presenting the state of the computer model to the user, and different ways of enabling the user to manipulate this state. In developing a computer model that represents a real-world state, the mode of presentation and manipulation is of the essence.

This conception of a database accords well with the original motivation for centralising data, as set out in [KS91], for instance. The first databases were repositories of knowledge about data that simplified the task of writing special-purpose programs for data manipulation to such an extent that in many cases they could be directly specified as queries by a naive user. In a modern context, we would like to develop a database in such a way that we can readily construct an customised *environment for interaction* for any agent requiring access, making data observable and manipulable within this environment in the ways that are most appropriate to that agent's real-world view of the data. For instance, in a children's library, it might be appropriate to respond to a textual query by presenting a picture of a book, stating its title and presenting a map representation of the library in which the child could simulate locating the book on the shelves by interacting in a standard video-game style.

1.2. The Impedance Mismatch: Application- or Machine-oriented?

A primary quality of relational database theory is that it exploits abstractions that are machine-independent. Fundamental ingredients of the data model –

such as the functional dependencies between attributes – are derived from the application, and used to inform the database design. A major obstacle to developing a satisfactory abstract framework for modern databases is that computer representation of real-world state has a **machine-dependent** aspect[2]. It is only when we take the view that an abstract relation can record the essence of a complex geometric object that we can treat the computer programs required to present such an object on a graphical display as a separate concern, outside the scope of the database model. If, on the other hand, we regard an abstract relation as an inappropriate metaphor for a geometric object, we have to give some suitably high-level and generic account of the program required to generate an appropriate display. This suggests that any satisfactory abstract model for a modern database will involve developing powerful principles for general-purpose programming.

The problem of integrating a programming notation with a good abstract data model is at the root of the well-known "impedance mismatch" to which Atkinson alludes in [A78]. Object-oriented principles are often favoured as a way of unifying application-oriented modelling and computer programming, but it is not clear that they can resolve the impedance mismatch without compromising the quality of the abstract data model. One reason for scepticism is that the primary function of the database – as we have identified it above – is to represent real-world *state*. The contents of a relation table and the image on a computer screen both represent an aspect of real-world state. In contrast, object-oriented programming offers convenient methods to specify *transformations of data*. The relationship between the application-oriented object-concept originally introduced in Simula, and the programming-oriented object-concept subsequently developed by Parnas and others, is also questionable.

2. Agent-oriented Modelling over Definitive Representations of State

The above discussion suggests that good principles for modern database design must be developed with very general modes of real-world simulation in mind. To be effective, such principles will have to take full account of the lessons that have been learnt from relational database theory, but cannot be framed exclusively in terms of tabular representations and relational operators. They will also have to supply programming environments as powerful as those developed using object-oriented principles.

The data representation methods we have been developing over recent years have good credentials for this role. They can be seen as a broad generalisation from principles that are well-illustrated in the design of the relational language ISBL [T76], and have very wide application beyond the scope of relational algebra. In our research programme, they have been successfully used to simulate a variety of physical processes, including a vehicle cruise controller, a game of billiards and the arrival and departure protocols at a railway station [BBY92, FBY93, BY92]. In section 3, we shall briefly consider their application to an exercise in concurrent engineering.

[2]. Our concern here is with the ways in which internal values are rendered perceptible to the user, not with "physical data independence" in the sense of [].

2.1. Observation and Experiment

Our mode of data representation is suggested by a conventional analysis of real-world systems through scientific observation and experiment. Each observable is represented by a variable, and the value of this variable is either defined explicitly or expressed by a formula in terms of the values of other observables. For instance, in measuring the extension E of a heated rod of length d with temperature T, we may formulate a definition

$$E = (1+\lambda T)\, d \qquad (+)$$

to express the way in which extension depends upon temperature and length.

Our perspective in adopting this representation is that of the experimentor rather than that of a theorist. Whilst the theorist regards E, T and d as mathematical variables constrained by a relationship, the experimentor regards the quantities E, T and d as genuine variables having an identity to which different values can be assigned. A definition thus expresses the latent propagation of change of value from one observable to another, rather than a constraint on a set of numerical quantities. For instance, (+) is interpreted as declaring how extension of the rod is subject to a change with temperature or length.

The representation of observables by variables in this fashion leads naturally to the use of a definitive script to represent system state. At any given point in the interaction with the computer model, the definition (+) is complemented by definitions for T and d. These might, in particular, define the current values of these parameters as measured by the experimentor. To interpret a definitive script, we must have some mode of interaction with the script in mind. For example, the experimentor may consider the extension of a rod with reference only to changes in its temperature, and consider the substitution of a rod of different length as *setting up a new experiment*. The central concept of representing data using definitive scripts is that we can accurately specify the way in which each particular agent perceives and manipulates data.

2.2. Definitive State-Transition Models

The identification of observables links our data representations closely to analysis of the application and supplies an implementation-independent ingredient to the data model. A definitive script can represent the state of a computer model at a suitable level of abstraction, subject only to interpretability of the operators that appear in defining formulae. When we formulate a script, we have in mind certain kinds of data value (to record the values of observables) and certain kinds of operator (to record the perceived relationships between observables). In the definition (+) the values are scalars, and the operators come from traditional arithmetic.

The data types and operators we exploit in constructing a definitive script make up the underlying algebra for our application. An ISBL view can be regarded as a definitive script in which the variables represent relations, and the operators in defining formula are drawn from relational algebra. The defining formulae of cells in a traditional spreadsheets describe a data model similar to a definitive script over standard arithmetic.

In building the computer model to represent the state of a real-world system, we aim to establish a correspondence – to be respected by all transformations that agent actions initiate – between values of observables and values of associated variables in the model. Note that this correspondence is between *states* of the real-world system (as defined, for instance, by the measured extension of a rod of a particular length at a particular temperature) and *states* of the computer model (as defined by the recorded values of E, T and d). To be consistent with the conception of a database that we have introduced above, we are led to consider how the current values of the stored variables E, T and d are presented to the user. Such a consideration determines the metaphor by which the computer model represents real-world state.

The metaphors available to us in making internal values perceptible to the user are highly machine-dependent. A system in which the value of E determined the frequency of a musical tone emitted by the computer would be an unusual choice in most contexts, but might be most appropriate where the effect of temperature upon the pitch of a percussion instrument was being assessed. In our approach to externalising values, we have concentrated largely upon developing visual metaphors. For instance, we can directly formulate definitive scripts whose variables represent geometric elements such as points and lines on a display, as in the definitive notation for line drawing DoNaLD, or represent content, location and attributes of windows, as in the definitive notation for screen layout SCOUT. The development of such notations allows us to exploit richer metaphors for interaction whilst retaining the same conceptual framework of realising and interpreting states using definitive scripts.

2.3. Agents

The representation of views of data to different agents is a major function of a conventional database. As we have illustrated above, the use of a definitive script to represent the state of a real-world system already embodies a supposition about how we expect to interact with the system. The definition

$$E = (1+\lambda T)\ d$$

is significant only in as much as we consider the possibility that d, T, λ or even the definition of E itself is subject to change. The different roles of agents in a real-world system are typically reflected in different privileges to observe and affect the values of observables.

In our approach, we have introduced a special notation LSD for specifying how each agent is privileged to observe variables and to change the state of the system through redefining variables. In an LSD specification, a variable that can be observed by an agent is identified as an **oracle** for the agent, and a variable whose value can be conditionally redefined is a **handle**. Each agent has certain privileges to change state: each privilege is represented by a guarded sequence of redefinitions of **handle** variables. The set of privileges of an agent together make up its **protocol**.

The specification of an agent in LSD resembles the specification of a class in an object-oriented programming language. In simulating system behaviour, agents can be dynamically instantiated and destroyed, and more than one instance of a particular agent can be active at the same time. During simulation, each observable in the system at any time corresponds to a variable whose very existence is linked to the presence of the particular agent that owns it. The set of observables owned by an agent in this way is identified as its set of **state** variables.

An LSD specification of a system is superficially similar to an object-oriented system such as the **actors** model [H77]. Agents that have only **state** variables fulfil the function of real-world objects, whilst agents with **handle**s resemble active objects. The major distinction between our agent-oriented model and a traditional object-oriented model is concerned with supplying the environment and scenario for interaction between agents that is required for operational interpretation. In moving to an operational interpretation, assumptions about the relationship between the authentic and observed values of variables have to be made: the framework for this interpretation is supplied by suitable definitive state-transition models. Part of this operational structure may be supplied by perceived indivisible relationships between observables, associated with **derivate** variables attached to agents.

The operational interpretation of an LSD specification is expressed using the Abstract Definitive Machine (ADM): an abstract computational model in which the state of an executing program, or evolving system, is represented by a definitive script D. Instantiation and deletion of agents leads to the introduction and deletion of groups of definitions from D. Within the framework of the ADM, the user has exceptional privileges to inspect and redefine the state of an executing program dynamically, in much the same way that a programmer operates when using a symbolic debugger. In concept, the ADM is a concurrent machine model, in which many redefinitions of variables in D can be performed simultaneously. Automatic detection of conflicts between such redefinitions (as when two agents simultaneously redefine the same variable, or redefine variables in a way that introduces cyclic dependency) is a powerful feature of the ADM model.

The scope for modelling agent interaction within the ADM model depends upon two mechanisms:

- parallel redefinition, whereby two (non-interfering) agent actions are performed simultaneously (cf parallel update of cells in a spreadsheet),
- redefinition in context, whereby one definitive script serves as a component of another to convey the way in which the effect of an action can propagate throughout a system in an indivisible fashion. For instance, definitions can be used to complement

  $$E = (1+\lambda T)\ d$$

  so as to introduce a listener who responds to changes in pitch of a vibrating rod that is being subjected to changes in temperature.

3. A Case-study: Concurrent Design of a Lathe Shaft

The method of data representation we exploit can be best illustrated with reference to a particular case-study that we are currently developing – the design of a lathe shaft as an exercise in concurrent engineering. The background to this study is described in greater detail in a previous paper [BC89]. In formalising the design task, we are led to consider the corporate action of four agents who liaise under the overall control of a design coordinator. The four agents we have identified are two engineering agents responsible for analysing the static and dynamic behaviour of the shaft respectively, and two agents concerned with detailing the form and choice of components and mode of manufacture respectively. In practice, the work of these four agents might be carried out by fewer than four people (cf Minsky's concept of the *society of mind* [M88]), whilst that of the design coordinator might be distributed amongst several parties in the design process. Be that as it may, the agent-oriented modelling principles we have described are expressive enough to enable us to conceive abstract agents and classify these as roles of actual designers as appropriate. In a commercial setting, many other agents would be involved, for instance, in the gear design, aesthetic design and detailed specification of manufacturing process.

The wealth of data representation issues involved in supporting the design task can be illustrated by analysing the roles of the design agents.

The static analysis of the behaviour of the lathe shaft under load requires an environment in which it is possible for a human agent to simulate locating loads on the shaft and changing the shaft profile by increasing its diameter where appropriate. For this purpose, it is convenient for the user to have a visual representation of the forces acting on the shaft and the deflection that it undergoes. A definitive script that sets up a "what if?" environment suitable for an engineer who wishes to explore the consequences of different hypotheses about the geometry of a shaft and the loading to which it is subjected is described in [BC89]. The graphical display associated with this script, as specified using the SCOUT and DoNaLD notations, is shown in Figure 1.

The observations associated with static analysis of the lathe shaft record

- the shaft geometry: the number and disposition of steps etc
- physical characteristics of the shaft characteristics
- the force locations: where the gears that rotate the lathe shaft and drive the cutting tool laterally are attached
- the responses to load

  shear, moment, deflection at all points of the shaft

  max deflection, max stress, stiffness (deflection / load).

The display in Figure 1 is the visualisation of the computer model of these observations. In static analysis of the lathe shaft, the engineer simulates the effects of experimentation with the loading and the shaft profile. The shear, moment and deflection are predicted from theory, and computed by numerical approximation and integration. There should only be a small discrepancy between observations as predicted by the computer model and those that would be observed in physical experiment.

The quality of the computer model depends on issues of implementation, such as the resolution of the display, the choice of step-size for the numerical computation, and the method chosen in displaying the graphs. The definitive script accessed by the engineer in static analysis includes such parameters to control the computer model.

The analyst concerned with the dynamic aspects of the lathe shaft performance is interested in other observables. These include

- characteristics of the cutting task
  workpiece characteristics: material, length, diameter
  choice of cutting tool, cutting tool profile
  back engagement (= depth of cut)
  side engagement (= lateral displacement per revolution)
- dynamics of the system
  forces on gears, power output of motor
  angular acceleration, speed of rotation
- factors influencing the quality of the product
  stiffness, maximum deflection at cutting point
  quality of finish, temperature

Notice how this set of observables refers to a context quite different from that conceived in static analysis. Dynamic analysis of the shaft involves making different choices of workpiece and cutting tool and simulating the behaviour of the lathe shaft in use. We have illustrated how agent-oriented modelling over definitive scripts can be used in simulation of this nature in previous papers [BBY92]. For this purpose, the agents to be identified and specified in LSD are the components of the engineering model, such as the cutting tool and the gears (cf [BBY92]). To make the transition from a static to a dynamic model, we introduce time via observation of a clock, and represent dynamically varying quantities, such as forces, by analog variables.

The engineering analysis of the lathe shaft is complementary to the work of the agent who details the parts to be used in its construction. Abstract information about component characteristics is used in conjunction with knowledge about the expected use of the lathe and the available components. We envisage that a detailer would ideally like at least two different kinds of view of the lathe shaft: an exploded and an assembled arrangement of the components. These two views are distinguished by the definitions that relate the location of components once assembled.

The definitive script developed for the detailer will incorporate a number of different components:

- object models
  Certain aspects of the lathe shaft design process, such as the choice of gears to suit function, are highly specialised. The detailer will typically accept the specifications of the gear designer as given. The component of the detailer's script that records gear attributes will incorporate definitions that the detailer cannot change arbitrarily (such as the

number of teeth, dimensions or constituent material), and others (such as the orientation of the gear on the shaft) that can be changed according to a prescribed pattern, as is required in simulation.

- catalogues

  In selecting gears to meet the specification of the gear designer, the detailer will refer to catalogues of parts. In our framework, a relational database with an ISBL-style interface is conceptually the most appropriate representation for such a catalogue.
- constraints

  The detailer's task involves reconciling conflicting demands, such as performance versus cost. We envisage two ways in which such constraints can be explicitly introduced into the definitive script:
  - boolean variables can be used to monitor constraints, and to flag failure to meet design objectives that are negotiable or temporarily unmet,
  - constraint enforcement agents can be introduced into the detailer's environment, so that a redefinition that leads to the violation of a constraint is prohibited.

The above discussion has outlined the observables and experiments that are used to specify the environments for various design agents. We should expect the design agents to use such environments to develop several design proposals concurrently, each constrained by different considerations. A shaft analyst who is used as a consultant by the detailer might be given specific details, such as the location of components on the lathe shaft. The same analyst may be independently involved in exploring a design in which the disposition of forces on the shaft is to be freely chosen. The definitive script only specifies the observational arena for experiment, and can be used in different ways according to what privileges for redefinition are presumed. In the concurrent engineering process, it is of the essence that the location of components can be influenced by many different parties.

The computational framework of the ADM is well-suited for representing the process of script generation and modification involved in concurrent engineering. Different versions of designs and partial designs can be represented as distinct instances of scripts of the same generic form. Different environments for script evolution can be set up by introducing appropriate sets of agents and characteristic scripts. These agents will typically include computational devices similar to the constraint enforcement agents that perform functions that ordinarily would be described as procedural programs.

The concurrent design process can be represented by concurrent execution within the ADM with conflict resolution supplied through the intervention of a design coordinator. The role of the design coordinator is to ascribe privileges to the design agents, to invoke agents where appropriate, to arbitrate where there is conflict between redefinitions, to introduce definitions and agents to monitor and enforce constraints and to preserve versions of the design or suspended design interactions.

4. Comparison with other Database Models

Our case-study illustrates how our approach in principle provides an appropriate framework for much of the computational activity and interaction associated with an engineering design database. Our agent-oriented abstractions can be applied to the database designer agent, to agents with an autonomous behaviour that are represented within the database, and to those who access the data as users. In this section, we shall briefly contrast our framework with other approaches to database design.

4.1. Agent-oriented vs Relational Database Models

The acyclic systems of data dependency expressed in definitive scripts resemble the data dependencies that underlie relational database design, in that both are derived from a consideration of how the values of observables are interrelated in the application. Both relation tables and definitive scripts organise observations according to functional dependency rather than by association with objects, and a relation may contain attributes of many different objects. The type of dependency recorded in a relation is restricted to what can be specified in a finite table, and the extensional part of such a table can be viewed as the explicit enumeration of function values. The dependencies represented in a definitive script are not necessarily defined by operators that can be tabulated in this fashion, and in general change dynamically as agents are created and destroyed.

As explained above, our methods of data representation can be seen as a generalisation of the principles used in the relational language ISBL [T76]. In practice, we have yet to exploit relational abstractions within our framework, and in general use definitive notations whose variables represent values at a lower level of abstraction than the set-level relational operators. Our case-study suggests that this is partly because the need to manipulate sets of observations emerges at a higher level of abstraction in the design process than we have considered in our previous research. Our experience in designing definitive notations shows that it is convenient to be able to specify complex data values both at the set-level and the component-level. Mechanisms to accomplish this are incorporated in the definitive notations DoNaLD and ARCA for interactive graphics. The general method used in ARCA is of most abstract interest in this context, as it uses definitive scripts to declare the mode of definition for variables, indicating at what level of abstraction a value is to be associated.

In practical terms, making allowance for its relative immaturity, our approach redresses the narrowness of the pure relational paradigm. The use of appropriate definitive notations and introduction of suitable agents enables us to deal in a systematic fashion with metaphors for data representation and manipulation beyond the scope of the table-based framework. There is a unifying method of querying the database of the same order of sophistication as the interrogation of values and specification of views in ISBL, if not always as user-friendly as that of SQL or QBE. We are able to represent objects with behaviour, to introduce processes dynamically and to express the effect of aggregation of objects upon their behaviour. Our approach also gives particular emphasis to the way in which different agents are privileged to access and modify data.

A more profound question concerns the theoretical foundation of our agent-oriented approach – a question to be examined in the next section.

4.2. Agent-oriented vs Deductive Database Models

A circumscribed set of data transformations, each of which can be expressed in terms of relational algebra, is sufficient to describe all the transactions of interest in many commercial data-processing applications. In this type of application, the relationship between the state of the real-world application and the computer model supplied by a relational database is most satisfactory, in that every real-world operation has a mathematically abstract counterpart. Date ([D90], p12) cites "the existence of a sound theoretical base" for relational theory that this mathematical abstraction allows as a major argument in its favour.

In applications where the data transformations can be circumscribed, it makes clear sense to make universal statements about relationships between data and to develop database paradigms based on logic. In other applications, such as the engineering design study considered above, the nature of the computer model required is less obvious. Of its essence, the process of design is open-ended and experimental and visits contexts in which the universal relationships between data are yet to be identified and the possible transformations of data are yet to be circumscribed.

The use of definitive scripts emphatically relates to activity that is as yet unformalised in the conventional meaning of the term. A script expresses the expectation of the experimentor who has yet to develop a theory. The computer models constructed in the ADM framework are interpreted state by state in relation to changes to external variables that have the quality of observables rather than static mathematical variables. When viewed as computations, they do not in general have an unambiguous operational interpretation – they lead to singular conditions where information is incomplete, or potentially inconsistent, and require the intervention of the user to supply missing information and resolve conflicts. The extent to which the process of modelling observations in the ADM can nonetheless be regarded as "formal" is discussed in [BR92]. What our experience has demonstrated is that the process of simulation using the ADM is a means to modelling a requirement in a precise and systematic fashion, and that this process can converge on a formal specification in much the same way that a body of experiments leads to the postulation of a theory (cf [BBY92]).

Many sophisticated databases incorporate triggers and rules whose effect can be superficially similar to the updating process in a definitive script. In as much as these mechanisms represent a propagation of changes in an indivisible manner ("at the speed of thought"), they resemble inferences, but their logical status is suspect. One advantage of our agent-oriented view is that we are better able to discriminate between different kinds of rule application, for instance, so as to avoid the pitfalls of non-terminating cyclic invocation of rules.

The use of definitions also relates to what is commonly viewed as the conversion of information to knowledge. For instance, there may be convention governing the useful life of a lathe shaft, as specified by loss of stiffness beyond a specified limit. The introduction of conventions of this nature, which involve indivisible propagation of change through human interpretation of observation, may be seen as introducing a knowledge-based element into the model.

4.3. Agent-oriented vs Object-oriented Database models

Object-oriented principles involve the association of observables by entity. We have argued that a good database paradigm will be associated with a mode of programming that supports real-world modelling and simulation effectively. By reputation, object-oriented programming is a powerful paradigm for this purpose. In practice, the association of observations by entity doesn't resolve several key issues of real-world modelling. For instance:

- an object-oriented approach makes it easy to maintain the consistency between patterns of observation associated with entities of a particular type, but real-world transformations involve indivisible interaction between observables associated with distinct entities (as for instance in a mechanical system such as the lathe). Object-oriented programming offers no general principles to guarantee consistency between observations across object boundaries. The problems of specifying the behaviour of aggregates of objects are relevant here [Bo93] – cf gathering together the components of an engineering model vs assembling them.
- objects have depth – the characteristics of an entity are determined by how it is observed. How an entity is perceived and how it can be transformed is a function of what agents are present in a system. The introduction of a new agent affects the observables associated with an entity and the methods of transformation that can be applied to it,
- observations may be associated with context supplied by more than one object: as when we consider the attributes of a relation.

The development of object-oriented principles has been influenced quite as much by programming concerns as by issues of real-world modelling. As a powerful programming paradigm, object-oriented programming helps us to construct computer models (e.g. allowing distributed modular development, leading to efficient implementations and potential for GUIs etc) but these models aren't necessarily a good abstract data model for the application. The separation between the essential and the accidental features of the data modelling exercise is hard to maintain.

The advantages of an object-oriented approach are most apparent in applications where there are many instances of the same abstract entity and it is convenient to have a ready-made computer model to match a circumscribed pattern of observations and transformations. This is particularly relevant to design processes that involve the selection of components whose functionality is determined and whose interaction with other components follows a preconceived pattern. Though the agent concept in LSD offers a means to generate many instances from a single template, it has a more elusive behaviour – the operation of an agent is not specified in isolation, but is determined by considering the whole system of agents with which it interacts, and making assumptions about the context for their interaction. The significant distinction between the two approaches is that object-oriented principles apply where local integrity of data is sufficient to guarantee global integrity, whilst our agent-oriented model considers data integrity only in a global setting.

Conclusion

We have developed a method of data representation that we believe deserves consideration as a possible basis for the future development of databases. Further work is required to explore the significant connections with existing database paradigms, to establish good theoretical foundations and to enhance the supporting software tools.

Acknowledgments

We are grateful to Valery Adzhiev for useful discussions.
We are also grateful to the SERC for support under grant GR/J13458.

References

[A78] Atkinson, M.P. *Programming Languages and Databases* Proc of 4th Int. Conf. on Very Large Databases, 408-419, 1978

[BR92] Beynon, W.M., Russ, S.B. *The Interpretation of States: a New Foundation for Computation*. CS RR#207, University of Warwick, January 1992

[BC92] Beynon, W.M., Cartwright, A.J. *Enhancing Interaction in Computer-Aided Design* Proc "Design and Automation" Conf., Hong Kong, August 1992

[BY92] Beynon, W.M., Yung, Y.P. *Agent-oriented Modelling for Discrete-Event Systems* IEE Colloquium on "Discrete-Event Dynamic Systems", Digest #1992/138, June 1992

[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.

[BC89] Beynon, W.M., Cartwright, A.J., *A definitive programming approach to the implementation of CAD software*, Intell. CAD Systems II: Implementation Issues, Springer Verlag 1989, 456-468

[Bo93] Bowers, D. *Some Principles for the Encapsulation of Behaviour of Aggregate Objects*, IEE Colloquium on "Recent Progress in Object Technology", Digest #1993/238, 1993

[Br91] Brown, A.W. *Object-oriented Databases: Applications in Software Engineering* McGraw-Hill International Series in Engineering, 1991

[FBY93] Farkas, M., Beynon, W.M., Yung, Y.P. *Agent-oriented Modelling for a Billiards Simulation*, to appear

[T76] Todd, S *The Peterlee Relational Test Vehicle – a System Overview*, IBM Systems Journal 15(4), 285-308, 1976

[M88] Meyer, B. *Object-oriented Software Construction* Prentice-Hall Int. Series in CS, 1988

[D90] Date, C.J. *Relational Database Writings* 1985-1989 Addison-Wesley

[DD92] Date, C.J. with Darwen, H. *Relational Database Writings* 1989-1991 Addison-Wesley

[H77] Hewitt, C. *Viewing Control Structures as Patterns of Passing Messages* Artificial Intelligence, 8, 323-364, 1977

[KS91] Korth, H.F., Silberschatz *Database System Concepts* McGraw-Hill 2nd ed. 1991

[M 88] Minsky, M. *The Society of Mind* Picador, London 1988

[W75] Wyvill, B. *An Interactive Graphics Language* PhD Thesis, Bradford University, 1975

Figure 1