CONSTRUCTIVIST COMPUTER SCIENCE EDUCATION RECONSTRUCTED

Meurig Beynon
Computer Science,
University of Warwick
Coventry CV4 7AL
wmb@dcs.warwick.ac.uk

ABSTRACT

The merits of Empirical Modelling (EM) principles and tools as a constructivist approach to computer science education are illustrated with reference to ways in which they have been used in teaching topics related to the standard computer science curriculum. The products of EM are interactive models – construals - that serve a sense-making role. Model-building proceeds in an incremental fashion through the construction of networks of definitions that reflect the observables, dependencies and agents associated with a current situation. The three principal case studies discussed (teaching bubblesort, solving Sudoku puzzles, and recognising groups from their abstract multiplication tables) highlight respects in which EM accounts for aspects of computing that cannot be effectively addressed by thinking primarily in terms of abstractions, procedures and mechanisms. The discussion of EM as a constructivist approach to computer science education is set in the context of an analysis of constructivism in computer science published by Ben-Ari in 2001. Reconciling EM's constructivist epistemology with this analysis involves recognising its pretensions to a broader view of computer science.

Keywords

Computer Science, Computer Science Education, Constructivism, Logic, Formality, Bubblesort, Sudoku, Group theory, Empirical Modelling, Observable, Dependency, Agency.

1. INTRODUCTION

In his paper “Constructivism in Computer Science Education” [1], Mordechai Ben-Ari draws attention to the way in which computer science education (CSE) must take account of the fact that “computers form an accessible ontological reality”. By this he means that, in effective interaction with computers, “a ‘correct’ answer is easily accessible, and moreover, successful performance requires that a normative model of this reality must be constructed”. This leads him to conclude that, whilst a constructivist pedagogical stance can be sustained in CSE, there is no place for “constructivist *epistemology*, which is nonfoundationalist and fallible". The basis for this conclusion is the uncontroversial observation that the behaviour of computers is by design reliable and predictable, and the syntax and semantics of programming languages is non-negotiable.

In reaching his conclusions about a constructivist approach to CSE, Ben-Ari implicitly subscribes to a particular stance on the nature of computer science itself. In this stance, programming stands at the core of the subject: the proper scope of computer science as a discipline – at least in so far as it can be deemed to be a science – is defined by how we can account for interactions with computers as of their essence "program-like".

When taken in this widely-accepted but narrow sense, computer science naturally places its fundamental emphasis on formality and formalism. As Ben-Ari observes, “intuitive models of computers are doomed to be non-viable” ([1],p56). This highlights the importance of being able to think in abstract mathematical/machine-like terms as a key skill for the computer scientist. It is in this spirit that Wing [2] champions “computational thinking”, that Kramer [3] poses the question: “Is abstraction the key to computing?” and that Dehnadi and Bornat [4] stress the importance in developing programming skills of being able to manipulate symbols strictly in accordance with rules without regard for their meaning.

There is a complementary story to be told. Few disciplines potentially engage more with every aspect of life, in all its breadth and informality, than computing. In many computing applications, such as games and music, it is the experience generated by computing technology that is most significant [5]. Coping with informality is crucial in developing complex software. Brooks [6] identifies the importance of seeking conceptual integrity, but the perspectives and sources that inform software are exceptionally rich and defy immediate formalisation. Software development accordingly is much concerned with hybrid activities that originate in a world of concrete experiences and potential confusion and strive towards rationalisation and order. This has been part of the motivation for broader approaches such as model-driven development, agile methods and participatory design.

In traditional approaches to marrying the formal with the informal, it is the formal nature of the computer that is in the foreground. Though few believe that formal methods and mathematical approaches to programming supply comprehensive solutions to the problems of practical computing, the idea that computing has its foundation in logic and mechanism is hardly ever questioned. Enlightened theorists nonetheless recognise the importance of the experiential aspects of computing. For instance, David Harel – a computer scientist whose thinking has strong theoretical roots – proposed statecharts as a way of giving greater expressive power to formalism [7], but in doing this acknowledged the importance of the visual in managing the complexity of the development process. His more recent research on "play-in scenarios" [8] also seeks to make direct links between the informal understanding of requirements that informs users' interaction with prototypes and refinements to the specifications of such prototypes. Jackson [9] wrestles with related concerns in his discussion of "What can we expect of program verification?", as does Ridley [10] in his discussion of "Database Systems or Database Theory".

The context to which Ben-Ari's characterisation of computers as an "accessible ontological reality" is most relevant is that in which classical computer science was originally conceived. In contemporary applications of computing technology, there are compelling reasons for seeking richer conceptual foundations for practice than a formal account of computing can supply [11]. The blurring of the boundary between computer science and engineering in many aspects of such practice itself presents a major challenge. When Ben-Ari ([1], p46) asserts that CSE "probably has more in common with engineering education than with science education", he acknowledges the potential limitations of his analysis of constructivism in CSE by inviting readers with an engineering background to speculate about its applicability to their field.

The modern context for software development typically embraces physical and human aspects in a way that has more in common with engineering than classical computer science. Where classical computer science is predicated on the prior completion of a design and engineering process that leads to the production of a programmable computer together with a generic abstract programming language, exploiting computing technology is now quite as much concerned with orchestrating the interaction between human and automatic agents communicating like the components of an engineering system as with programming individual devices. In aspiration, this orchestration leads to a product that is sufficiently fluid to allow for negotiated requirements and emergent uses, in keeping with the spirit of a constructivist epistemology such as Ben-Ari recognises as inappropriate in the natural habitats of classical computer science. The difficulty of supporting such negotiations of meaning and realising such flexibility in practice with methodologies that are rooted in the epistemological framework of classical computer science is acknowledged by Kallinikos in [12].

In this paper, Empirical Modelling (EM) is introduced as an alternative approach to computing that embraces a constructivist epistemology. As explained in more detail in [13], EM can be seen as a generalised form of programming, but one that incorporates both the identification of reliable mechanisms and patterns of interaction that must be carried out prior to the specification of processes in orchestrating the development environment and the sense-making activities that bind state-change to meaningful entities that can be observed and manipulated appropriately by the human and automatic agents involved. In keeping with Cantwell-Smith's analysis of the role of logic in computing applications [14], the target for this modelling activity is not the semantics of the programming language, which relates merely to the abstract computational mechanisms that are being specified, but what Cantwell-Smith describes as the 'semantics of the semantics'. That is to say, as will be illustrated in the next section, the modelling activity is directly guided, moment-by-moment, by the desire to craft the current state and potential for atomic state-transition so that it is congruent to the current state and potential for atomic state-transition in the referent.

As an approach to computer science education, EM is based on foregrounding the informal and experiential aspects of interaction with computing technology. The spreadsheet is a helpful motivating example of an application that embraces these aspects. A spreadsheet that functions effectively relies upon the computer to update dependencies in a timely fashion, and display the results to the human interpreter. Though the algorithms that effect the dependency maintenance are formally specified as far as their abstract function is concerned, it is necessary to invoke real-time concerns – and in general to take the characteristics of the human interpreter into account – to determine whether these algorithms execute at an appropriate speed. What is more, because the complexity of the dependency networks that need to be updated is determined by the external meaning of the values in cells, it cannot in general be constrained to a fixed size or depth. On that basis, the most important characteristic of the spreadsheet relates to informal, potentially observer-dependent and hardware-dependent semantics that cannot be comprehensively formally specified.

This paper reviews some applications of EM in computer science teaching, highlighting issues that prove problematic when considered primarily from a classical computer science perspective. Section 2 is a brief introduction to the principles and tools of EM. Section 3 illustrates, by means of a simple example (viz. Teaching Bubblesort), how EM can be used to explore the kinds of experimental activity and empirical

knowledge that lie behind the identification of algorithmic procedures. Section 4 considers how elaboration of the EM bubblesort model exposes the subtlety of the relationship between processes that admit formal analysis in the spirit of theoretical computer science, and those that demand a more pragmatic “engineering” approach. Section 5 discusses the way in which EM shifts the focus from using the computer as a calculator to disposing the results of calculation semi-automatically in such a way that they can be more readily assimilated and experienced as meaningful. This is illustrated in Section 6 with reference to EM construals to support human solving of Sudoku puzzles and the development of an environment in which the five distinct abstract mathematical groups of order 8 can be displayed so as to expose their structure and informal interpretations.

The four EM environments discussed in this paper (relating to a visual pun, bubblesort, Sudoku and group theory) have all been developed using variants of the principal EM tool: the EDEN interpreter (see the Software link on the EM website [15]). All four can be accessed using the web-enabled variant of EDEN that has been developed by Myers [16]. By exercising these Web Eden models, interested readers can trace the interactions discussed in this paper themselves. This is helpful, if not essential, in gaining a good understanding.

2. EMPIRICAL MODELLING PRINCIPLES AND TOOLS

Empirical Modelling (EM) is an approach to computing based on principles and tools that have some key qualities in common with spreadsheets (see [15,17] for more background). The primary concepts in EM are observables, dependencies and agents. An observable more closely resembles the quantity associated with a spreadsheet cell than a traditional program variable, in that it refers to some significant quantity that is meaningful in the external context. (For instance, a cell may relate to the mark achieved by a particular student on a specific exam, or to the average mark attained by all students on an exam etc.) The definitions that relate the cells of a spreadsheet express dependencies between the values of observables such that a change to the value of one observable effects changes to the values of other observables according to some formulaic recipe (such as is used to compute the average mark for an exam). The ways in which the cells of a spreadsheet are liable to be updated are linked to the kinds of agent that are privileged to redefine the values or defining formulae of cells. (For instance, each mark associated with a specific exam is assigned by the examiner, whilst the way in which the overall mark for a module is computed from components of assessment may be subject to modification by the examination board to take account of special circumstances.)

EM has been under development in Computer Science at the University of Warwick for many years. Over that period, it has been the topic for a distinctive contribution to the curriculum at the undergraduate/postgraduate interface. Its potential as an educational technology has been the theme of many papers (see e.g. [18,19,20,21]) and of two doctoral theses [22,23]. Much understanding has stemmed from making links between EM and mainstream computer science – see e.g. [21,24,25]. EM principles and tools have also been employed to support teaching of traditional computer science. Relevant resources, all of which are available for download from the EM archive at [26], include models of the modes of the visual editor "vi" (vimodesBeynon2006 at [26]), of the properties of projection mappings in computer graphics (graphicspresHarfield2007 at [26]), of the display and mechanisms of a digital watch (digitalwatchFischer1999 at [26]), of the heapsorting algorithm and an associated weakest precondition specification (heapsortextendRun-bol2001 at [26]), and of the evaluation of SQL queries in a relational algebra framework (sqleddiWard2003 at [26]). Some of these exploit the EM presentation environment (empeHarfield2007 at [26]) in which all aspects of the model and presentation are expressed by dependencies maintained by EDEN (cf. Figure 1).

A full elaboration of the ideas and concepts of EM is beyond the scope of this paper; a brief discussion is sufficient for the present purpose. In EM, an observable is an identifiable entity to which a status or value can be associated in a particular context. In the first instance, the concept refers to elements that contribute to the modeller's direct experience of the current situation. By extension, an observable might relate to the role of a human agent other than the modeller, such as the teacher or learner in the context of an educational model. An observable might also describe some aspect of a situation which a non-human agent could register and respond to immediately, but which a human agent might not be able to access or react to in the same way. Observables relate to what can be deemed to directly influence the behaviour of state-changing agents.

The objective in framing observables and agents is to express plausible expectations and explanations for the way in which a situation changes, either as a result of intervention on the part of the modeller or autonomously. This we call 'making a construal'. An observable corresponds to "what can potentially undergo change", and an agent is "what is deemed to be responsible for change". A dependency then expresses the way in which changes to observables are deemed to be linked – if a change to an observable x directly effects changes to other observables in some predictable way, these observables are said to depend on x. The pragmatic nature of the characterisation of observables, agents and dependencies reflects the many different construals we can make of one and the same situation.

The model cabinetdigitpresBeynon2007 (available at [26]) introduces a simple example of an EM construal. Screenshots from the model are presented in Figures 1a and 1b. To the left-hand side of these screenshots there is an Interactive Display that shows the current status of some of the key observables in the model and an Input Box through which new definitions of observables can be introduced.

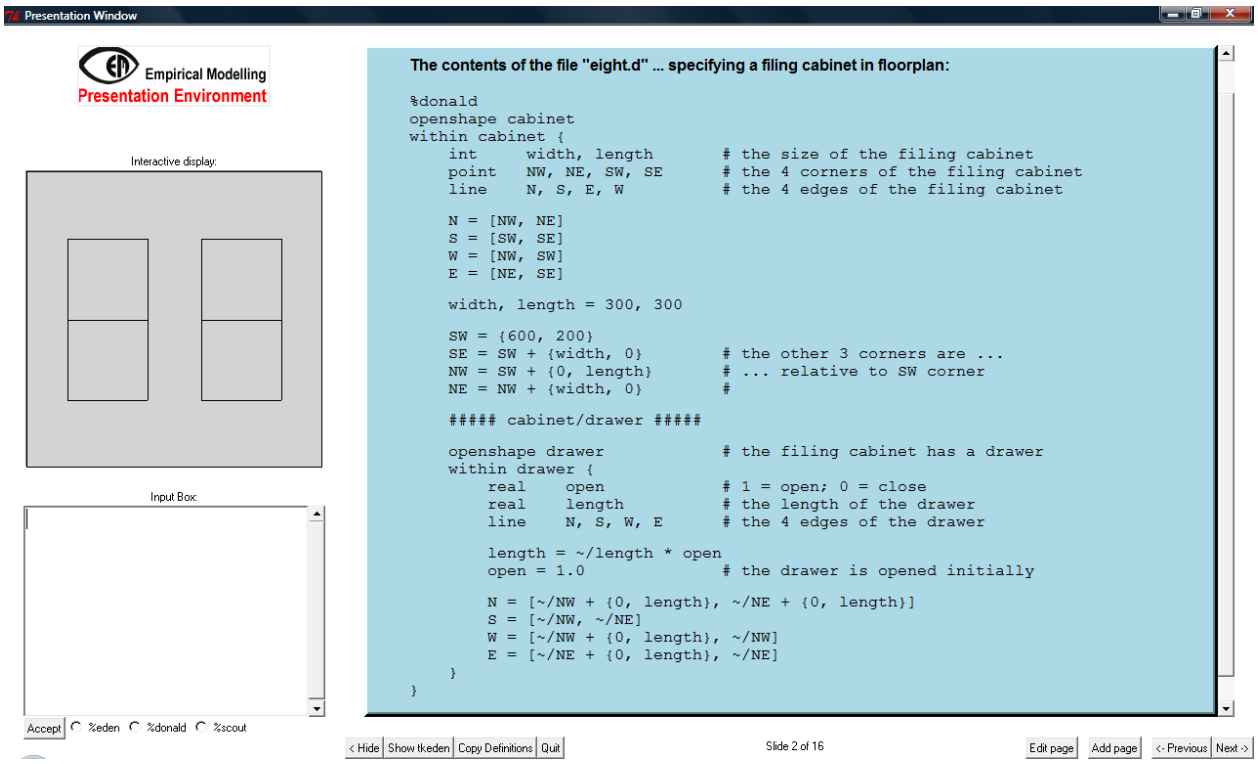

Figure 1a: A screenshot depicting an open filing cabinet in plan, together with its associated script

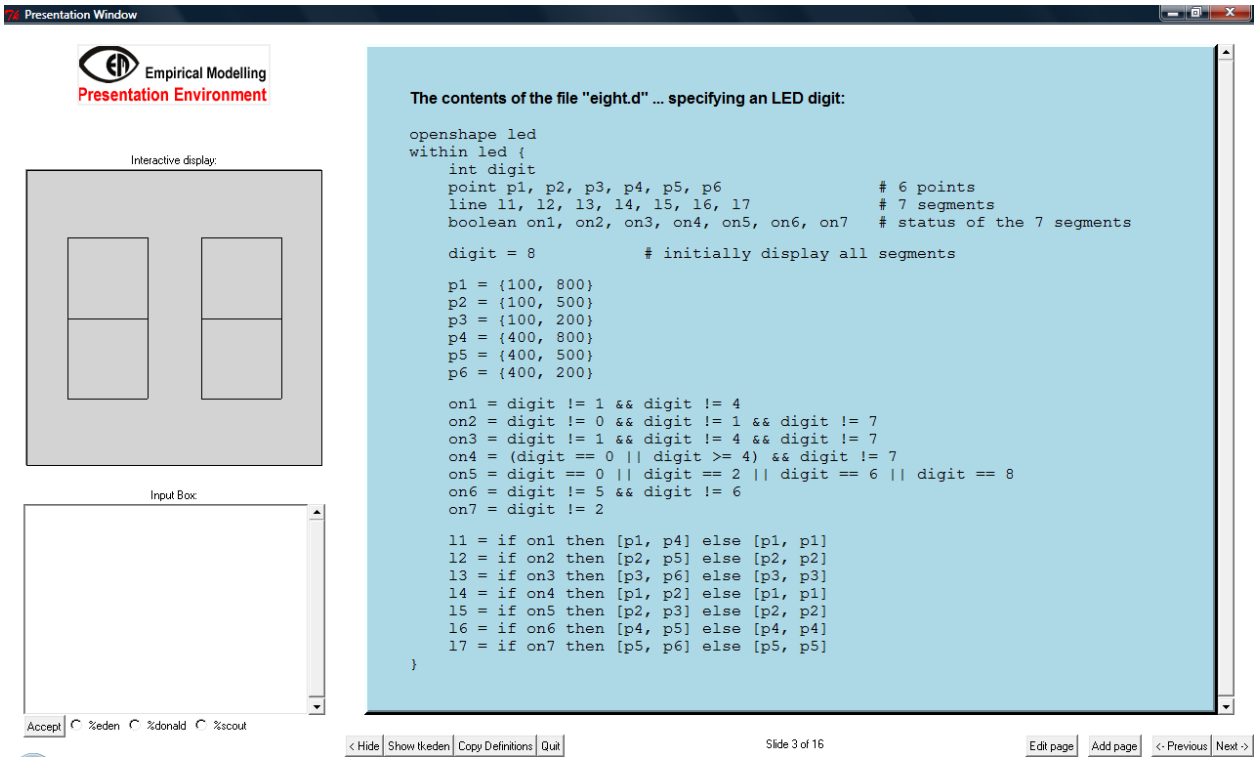

Figure 1b: A screenshot depicting an LED displaying the digit 8, together with its associated script

The two identical line drawings that feature in the Interactive Display constitute a visual pun. The set of definitions, or *script*, that defines one of these drawings is displayed in the panel to the right hand side of Figure 1a. This drawing can be construed as the floorplan of a filing cabinet that is currently fully open. The dependencies in the script are such that if the observable **open** is redefined to a value between 0 and 1, the associated line drawing is reconfigured so as to represent a filing cabinet that is open to a degree between 0 (closed) and 1 (fully open). Such a redefinition can be made by entering a new definition for the observable **open** via the Input Box.

The script that defines the second line drawing is depicted in Figure 1b. In this case, the line drawing is to be interpreted as a digit from an LED display – currently as the digit '8'. Redefining the value of the observable **digit** to a value between 0 and 9 transforms the corresponding line drawing so that it displays that digit.

Conceptually, an EM construal is to be thought of as an interactive environment rather than a program. At any point in interacting with the construal, there is an associated current family of definitions similar in character to those that link the cells of a spreadsheet. Though these definitions specify a dynamic updating activity ("dependency maintenance") that is carried out automatically by the EDEN tool, this is invisible to the human interpreter. The definitions are rather to be regarded as "defining the current state". In interaction with the construal, the modeller (or a human agent in some other role) explores her construal through redefining the current state in a thoughtful manner. The trajectory for this exploration need be in no way preconceived or fixed. A useful analogy may be made with taking a walk, where we are not obliged to place our feet in specific places, are at liberty to make digressions or retrace our steps, and may or may not be led by reference to a guide, a guidebook, or our previous memories.

The character of the knowledge that is associated with interaction with a construal is quite distinctive. Interaction with the construal serves as a means to disclose and challenge understanding. As a simple illustration, as depicted in Figure 2, redefining the observables **open** and **digit** disambiguates the identical line drawings displayed in Figure 1.

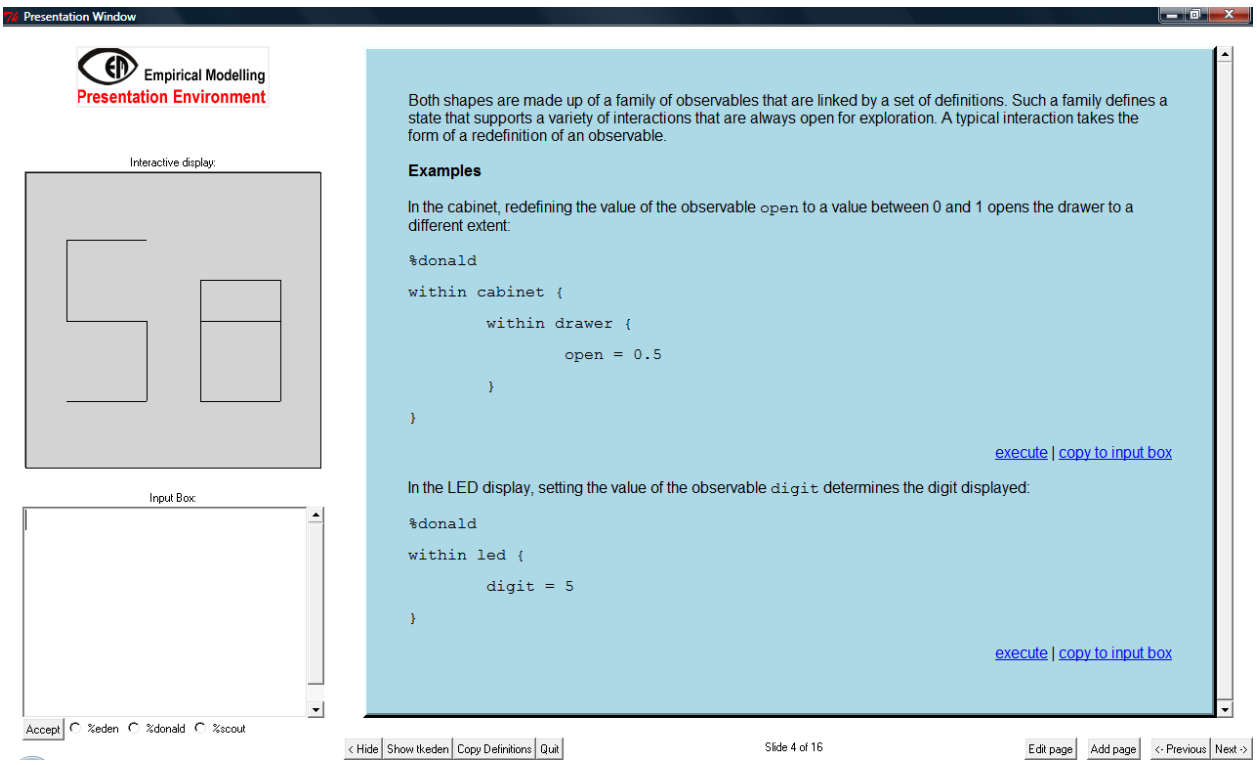

Figure 2: Disambiguating the two identical line drawings through redefining key observables

In general, EM construals have to be interpreted in conjunction with patterns of interaction and interpretation that have been rehearsed and become familiar. The meanings of the observables are expressed only in this way, and remain at all times fluid and negotiable. Closer examination of the dependencies that define the filing cabinet drawer in Figure 1a reveal that the lines that define the drawer do not in fact behave like a real drawer – as the drawer is closed its projection on to the floorplan is squashed flat to the cabinet. Though this suits the purpose of the visual pun, it is unsatisfactory when we attempt to "pull the drawer right out" by setting **open** to a value greater than 1, since this simply "stretches the drawer". One natural way to adapt the model through

redefinition is to reconfigure the dependencies that connect the lines that describe the drawer to the observable **open** so as to address this problem.

The way in which an EM construal 'represents' its real-world referent is quite different from a formal representation. How changes to observables are linked in interactions (like "opening the drawer") that are perceived as atomic is crucial to the interpretation of the cabinet and the digit. A whole variety of atomic interactions are afforded by the families of observables and dependencies associated with the cabinet and the digit. They form an open-ended family of possible redefinitions some of which are evidently meaningful, some of which are interpretable with a little imagination about the possible context, and others (the overwhelming majority) that appear meaningless. Instances of redefinitions might be: making the width of the cabinet or the LED very small, or making two vertices of the cabinet plan or LED coincident. The negotiation of meaning in this context is in tune with a constructivist epistemology as characterised by Ben-Ari in [1]. It is also oriented towards an engineering perspective rather than a classical computer science perspective on the problems of system development. The relevance of this for education is endorsed by Ferguson's observation in [27], p168:

"The real problem of engineering education is the implicit acceptance of the notion that high-status analytical courses are superior to those that encourage the student to develop an intuitive 'feel' for the incalculable complexity of engineering practice in the real world."

The plethora of atomic interactions afforded by the scripts in Figure 1 and the extent to which each of these is interpretable reflects the "incalculable complexity" of the referents associated with these geometric figures.

In the elaboration of an EM construal, a significant feature is the neutrality in the kinds of agency and interpretation that is being invoked at all times. There need be no specific user or purpose in mind and the modeler can freely elide from one role – such as "demonstrating the visual pun" in Figure 1, to another – such as "making the drawer more realistic". In a similar spirit, in the modeller's mind, the LED and the cabinet can be seen as disjoint or as conjoined, just as in everyday experience I may apprehend two distinct objects in one scene either separately or together. In the context of Figure 1, such a connection between the digit and the cabinet drawings can be made by introducing a dependency that opens the drawer to a degree that is determined by a digit between 0 and 9 for instance. It is also possible to invoke automatic agency to enable specific kinds of redefinition (e.g. to add a slider to set the value of **open**) or to animate specific patterns of redefinition (e.g. progressively opening the drawer by incrementing **digit** from 0 to 9). In this fashion, more constrained and program-like behaviours can be enacted within the model, though the essential openness to intervention and redirection through interaction in any intermediate state remains.

The neutrality of the agency in a construal means that EM principles are particularly well-oriented towards teaching and learning in a constructivist idiom [18,19,21,22]. In the constructivist ideal, learning, teaching and development have to blend in a manner that is ill-supported by traditional programming. By exploiting construals, the roles of a developer, a teacher and a learner can all be identified with different patterns of interaction and interpretation, each of which involves different varieties of redefinition associated with one and the same model of state. Subject to making the process of redefinition more accessible to the computing non-specialist, it is possible to imagine that an EDEN-like tool, especially if web-enabled, will support a rich process of distributed development, whereby teachers and pupils elaborate their own extensions to learning environments and make these available to others. The potential for integrating Web Eden with established learning environments, such as Moodle, has already been demonstrated by Harfield [20].

3. TEACHING BUBBLESORT

Bubblesort is an example of the kind of elementary algorithm that can be used to introduce basic procedural thinking. The application of the principles and tools of Empirical Modelling (EM) to CSE is introduced here by showing how they can be used to construct a model to support teaching bubblesort (see Figure 3 and [16]).

The key concepts in EM are *observables*, *dependencies* and *agents*. Making the Bubblesort model involves identifying how these concepts relate to teaching bubblesort, and – through an incremental process - embodying these concepts in the model. The radical difference in emphasis between making such a model and writing a conventional computer program is that the focus is placed on what the learner has to experience in order to understand the principles of bubblesort as they might be taught using old-fashioned teaching techniques, such as using chalk on a blackboard. In the first instance, the Bubblesort model aims to produce a teaching and learning experience similar in character to blackboard use, but one that exploits the power of the computer to enable change and maintain relationships in ways that could not be directly supported without such a technology.

The discussion in Section 2 sets the scene for model-building in EM. Whereas programming is of its essence concerned with implementing a process (as e.g. in a computer animation of bubblesort), EM is primarily concerned with building an interactive artefact (cf. the array that is drawn on the blackboard) whose current

state at any time reflects that of a current situation (cf. the specific set of values presently in the array) and context (for instance, whether or not a bubblesort is currently in progress). The reference here to 'current state' is an acknowledgement that the artefact, like the drawing on the blackboard, is to be interpreted by taking into account interactions with it in which human thought processes and supplementary communication play an indispensable role. This highlights a fundamental difficulty – no matter what technology is available – in conveying a sophisticated computer science concept, such as a *procedure*, a *computational step* or an *optimisation*; what is there to be experienced by the learner at any moment, as in a snapshot of current state, is a mere hint at what is in the teacher's imagination. It is in order to convey the richer picture that the teacher interacts with the artefact on the blackboard, modifying it so as to draw attention to the range and significance of possible contexts and issues this interaction can conjure in the learner's imagination. EM aspires to support a similar kind of open-ended interactive and interpretative activity.

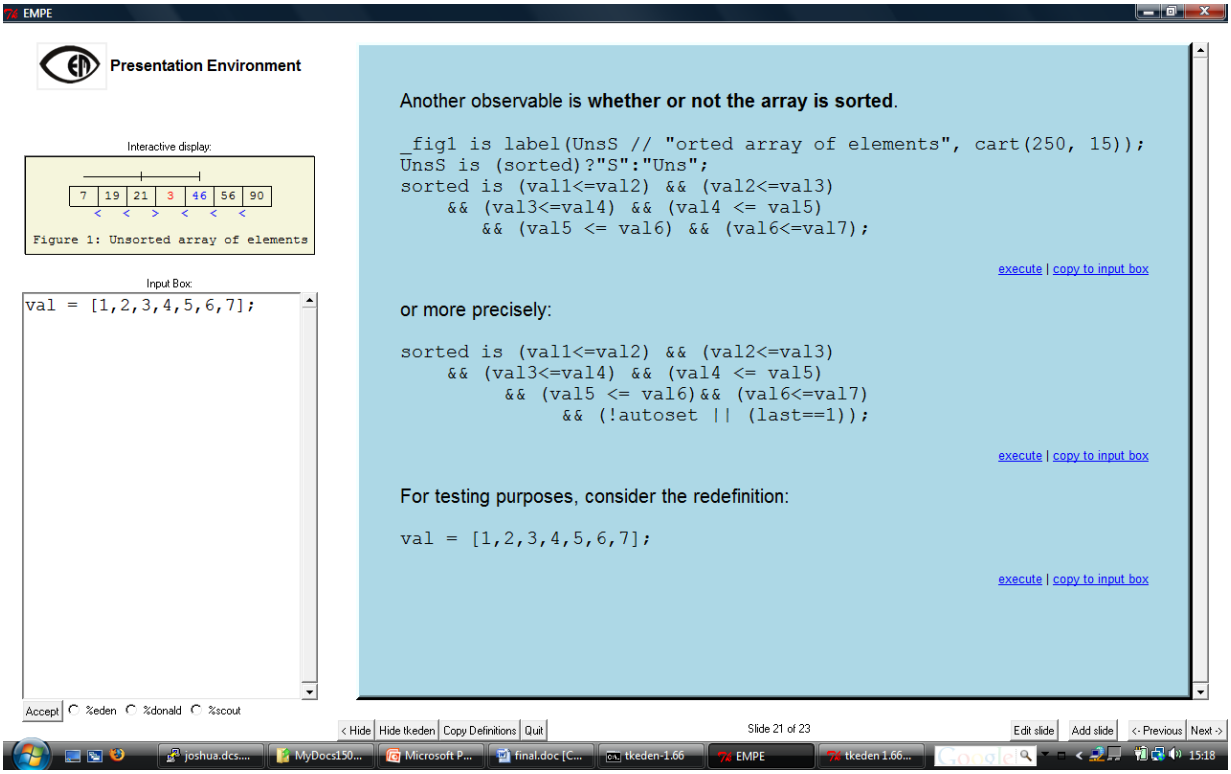

Figure 3: An EM construal for bubblesort presented using the EM presentation environment

Note that the primary emphasis in this approach to teaching an algorithm is necessarily on illustration by a specific example, though of course the teacher may make exploratory adaptations in an open-ended interactive way. A teacher might draw an array on the board for instance, and modify and annotate it in conjunction with asking certain questions about it: "Let's suppose that we are sorting the array of numbers <3,2,23,4,64,1,18>"; "Focus your attention on the biggest element in the array"; "What would happen if two numbers were identical?"; "What pass in the bubblesort process is currently in progress?"; "What is the next step in the bubblesort?"; "What would happen if the array was already sorted?"; "Would the process work if one of the numbers in the array was changed in the middle of the sorting process?" etc. Though these questions may seem rather concrete and elementary in nature, they involve quite sophisticated implicit assumptions about the context in which a question is being asked that are not normally given explicit support in a computer-based learning environment. For instance, some relate to the properties of the specific array as a static object prior to any sort being in progress, others to the condition of the partially sorted array when a bubblesort is in progress, others to the relationship between the bubblesort process and possible instances of arrays to which it might be applied. Note also that some informative questions might not strictly speaking relate to bubblesort at all – for instance, the possibility that array elements might be changed during the sorting process is not admitted in the formal problem specification.

Superficially, a computer animation has advantages over the chalk-and-blackboard approach where distinguishing between different contexts of interpretation are concerned. The computer can display the static array to be sorted, and display the sequence of states through which the array passes in bubblesorting. From the learner's perspective, it is important to note that – whichever approach is being used and whatever the context for the observation – what is visible at any particular moment is merely an array of numbers in a particular state. And whilst the computer animation may make provision for 'contemplating the current array as

a static object' and 'contemplating the current state of the array whilst bubblesort is in progress', these two possible contexts for observation of an array are much less nuanced than the contexts to which the teacher can draw attention at the blackboard. The important distinction being made here is between merely enabling the learner to see the effects of the sorting process in operation and helping the learner to imagine the mechanisms by which this process is guided (cf. [28]). It is learning of the latter kind that is promoted by the focus on identifying observables, dependencies and agents in EM.

Identifying the observables involved is the first step in building a model to assist in teaching bubblesort. In teaching bubblesort using a blackboard, the learner's attention is drawn to the physical representation of an array on the board, and to the values placed in this array. Because of the elementary nature of the sorting process, the only kind of comparison and exchange to be considered is that of elements that are adjacent in the array. When the basic steps of comparing and exchanging consecutive array entries have been introduced, attention can be given to the way in which these steps are combined in a systematic way in the sorting process. It is natural for a teacher to first demonstrate this by hand-tracing the sorting process and highlighting the way in which it can be described with reference to several passes in which each pair of adjacent entries not yet output from the sort is considered in turn. In the context of such a trace, the *index of the current pass* is a meaningful observable.

The most appropriate choice of construal in general depends on the teaching context. For instance, the description of the sorting problem in terms of the observables identified in the above discussion is only appropriate if the learner is able to interpret the entries in the array and immediately assess which of two entries is the bigger. If this level of familiarity cannot be assumed, the analysis of observation required would be yet more detailed. It might take account for instance of how individual digits were being observed and interpreted in order to compare array entries. Modelling this component of understanding might itself be the subject of another similar study.

In a blackboard based exposition of bubblesort, a teacher may well wish to perform actions outside the scope of the formal algorithm. They might change a value in the array in the middle of the sorting process, for instance, to demonstrate some feature of the sort. Such on-the-fly reinterpretations of context underline the richness of the construals that come into play in teaching an algorithm, but are not well-matched to the interpretation that surrounds a typical computer implementation.

4. RELATING THE FORMAL AND PRAGMATIC PERSPECTIVES

The distinction between the tightly prescribed formal computer algorithm and the teacher's open-ended informal construals illustrates the relationship between formal and pragmatic perspectives on computing in microcosm. The invariants that guarantee the correctness of the algorithm cannot apply whilst the teacher is liable to intervene in the quasi-computational process in an arbitrary way. It is also significant that the interventions that subvert the orthodox sorting process are those that have particular educational value, as they alert the learner to important assumptions that are tacit in routine executions. For instance, as further discussed below, it is helpful to highlight the extent to which the bubblesort algorithm can cope with on-the-fly changes to the values being sorted, as this gives insight into the character of the invariants.

The EM construal of bubblesort is a vehicle for practical exploration of the way in which conflicts between logical constraints on state change and open-ended agency can be resolved. As illustrated in section 2, the modeller at all times has the power to change the state of the construal in ways that defy prediction and expectation, so that anything that may be asserted by way of a predicate governing the relationships between observables is liable to be falsified. Certain observables, such as *the current pass in the bubblesorting process*, are meaningful only provided that the state-changes to which the model is subject are constrained. Logical concepts, such as "invariants of bubblesort", are subject to similar considerations. The term 'constrained' may here relate to discretion on the part of human agents ("the modeller chooses not to redefine values in the array"), limitations on action imposed by the interface ("no scope for inappropriate redefinition is afforded by the interface to a human agent acting in a specific role"), or automated agency that respects the integrity of the model ("the appropriate sequence of comparisons and exchanges is programmed to occur").

By way of illustration, it may be asserted of a bubblesort of n elements that, in the k-th pass (where $1 \leq k \leq n$) the elements with indices between n-k+1 and n are in sorted order, and, for $k \geq 2$, the element with index n-k+2 is greater or equal to every element with index between 1 and n-k+1. Such an invariant can be treated as a form of observable. For this purpose, a boolean observable whose value is defined by the appropriate predicate can be introduced to the model. The current value of this observable depends on the current pass of bubblesort and on the current values of the entries in the array. The value of this observable will always be true when interaction with the bubblesort model respects the steps of the bubblesort algorithm. Viewed in this way, the observable is an entity whose value is constant subject to exercising discretion in interacting with the bubblesort model and interpreting this interaction.

From a teaching and learning perspective, it is the nature of the constraints that an invariant places upon the sorting process that give most insight into its meaning. Though in fact the invariant is never violated in following the bubblesorting procedure faithfully, the teacher may well find it appropriate to draw attention to circumstances in which the invariant *might* be violated, as this obliges the learner to study the invariant more carefully and highlights characteristics of bubblesort. For instance, it is apparent from the invariant that permuting the values of array elements that have yet to be output (i.e. whose index is in the range 1 to n-k+1, where k is the current pass) and that have yet to be processed in the current pass (i.e. that have yet to be compared with another element during the current pass), will not subvert the sorting process. More generally, modifying these values in any way that renders them no bigger than the element with index n-k+2 can be accommodated. The fact that normally an additional invariant is imposed upon bubblesort to ensure that the set of values being sorted does not change in the course of the sort does not detract from the educational value of contemplating this type of experimental intervention in the normal sorting procedure.

A natural extension of this kind of activity concerns sorting a set of keys that are subject to change through the dynamic intervention of agents. As a simple example, it might be that bubblesorting is being conducted in a context where the value of each key is being incremented at each tick of a clock. Because the invariant discussed above refers only to the relative size of keys, it is apparent that this does not subvert the sorting process. If on the other hand some limits are placed on the size of keys, so that the keys are indices modulo a specific integer for instance, then the process of incrementing keys from time to time leads to the value of a key jumping from being the largest to the smallest. Depending upon precisely at what stage this jump in value occurs, and which keys it affects, this may or may not lead to violation of the invariant. It is clear that reference to the invariant then provides the information required to restore the bubblesorting process to a consistent point, though this may involve revisiting an earlier pass.

The EM model of bubblesort is an appropriate experimental testbed on which to explore such variants of bubblesort (see [16] for a Web EDEN implementation). In considering the possible behaviours that arise when considering how keys may be affected by agents acting in parallel, possibly in ways more random than those proposed above, it becomes clear that – useful as invariants may be as auxiliary observables in guiding recovery procedures – engineering issues become quite as prominent as abstract logical concerns. It would be impossible to make a comprehensive analysis of possible quasi-bubblesort behaviours under different circumstances without considering both the speed and frequency with which agent interventions were prone to occur and the speed with which the steps of the sorting process itself might be executed. If a variant of bubblesort were being used to maintain the order of a set of keys that was subject to very rapid updates (e.g. as a result of real-time update of sensory data), there would be intervals of time during which the invariant was invalid and the keys were unsorted, and just how far this would subvert the sorting process could not be predicted without both experimental and analytical study of how the steps in the updating and sorting processes were synchronised. And whilst some computer scientists may wish that the study of such real-time concerns might be guided by a more theoretical ("computer science") and less pragmatic ("engineering") approach (see for instance [29] p.16), it seems implausible that any general theory can be invoked unless specific and artificial constraints upon the nature of the agency involved are presumed.

5. FROM CALCULATION TO DISPOSITION

The core focus of computer science on calculation and computational processes is a natural legacy of history. It promotes a valuable emphasis on formalising activities to the point where they can be automated. But it is also essential to give a proper complementary account of the exploratory human activities that precede and potentially defy automation. To neglect these is to endorse a false impression of the relationship between reasoning and sense-making. EM redresses the balance between automation and exploration by developing principles and tools that can support sense-making. This is relevant not only prior to rationalisation and automation but also in comprehending and assimilating the results of computation.

The spreadsheet and the statechart exemplify the kind of support that the computer can offer in sense-making, especially where artefacts of this nature are being developed and used in a dynamic interactive fashion. Such uses are documented for instance in Baker and Sugden's review of applications of spreadsheets in education [30], Nardi's discussions of the use of spreadsheets in software development [31], and Horrocks's account of the benefits of using statecharts in maintaining interfaces [32].

The role that computer artefacts such as spreadsheets and statecharts play in mediating understanding and communication owes at least as much to the way in which they impact on human cognition as to their abstract mathematical content. It is *how* the results of calculations and computationally significant relationships are *disposed* and mediated to the human interpreter that gives these artefacts special power to support human processes. Paying primary attention to disposition rather than to conceiving procedures to meet specific goals

is characteristic of a culturally quite distinct tradition of thought such as is represented in the ancient Chinese notion of *shi* [33].

EM is particularly well-suited to giving computer support for exploring dispositions [34]. Embodying patterns of observables and dependencies in artefacts and subjecting these to open-ended interaction and interpretation reflecting many different kinds of agency is a means to comprehending a domain. As has been illustrated in previous sections, EM can generate construals that can then be applied in many different ways in support of both manual and computational processes.

The above discussion has relevance for teaching basic mathematics for computer science. In introductory mathematical modules to support core theoretical topics such as formal specification and verification, automata theory and abstract data structures and algorithms, it is tempting to emphasise the formal aspects of mathematics that are best matched to automatic calculation and inference. For instance, it is easy to give disproportionate emphasis to the automatable procedures that have been developed in classical mathematics, and to mathematical structures as exemplifying logical theories based on axioms and inference.

To exaggerate the role of mechanical deduction is of course to misrepresent the nature of mathematics as it is practised [35]. Computer science students are often alienated by a vision of mathematics of the kind that might appeal to a robot rather than a person. Without appreciating the roots of mathematics in human experience of problem-solving across a wide spectrum of applications, they cannot fully appreciate the benefits and power of mathematical abstraction.

Mathematics does indeed highlight many instances of problems that can be addressed through introducing powerful abstractions and techniques. In this respect, it provides inspiration for the application of formal methods to the task of developing complex software systems. But formalised mathematics did not precede the informal exploration of structures and procedures, often carried out in response to "ambiguity, contradiction and paradox", that Byers [35] identifies as a vital component of the discipline. It is explorations of this nature that led mathematicians to formulate abstract logical systems of axioms and inference [35]. Similar kinds of activities are associated with what Brooks [6] characterises as establishing "conceptual integrity" in software development. And though many mathematical problem contexts and problem-solving procedures have been rationalised to the point where axiomatisation and inference can be invoked, they do not in general address the problems that the software engineer faces in conceiving a complex new application.

6. CASE STUDIES IN CONSTRUING MATHEMATICS

Two case studies serve to illustrate how EM construals can engage with the creative and intuitive aspects of the mathematical agenda. The first is represented by a series of workshops, originally developed for gifted and talented school pupils with an interest in computing, devoted to construals that support the human solver of Sudoku puzzles (see [16]). The second is an environment, developed in conjunction with teaching a second year module on formal specification and verification, that shows how the tools of formal methods can be used to generate mathematical structures and how an experienced mathematician might interpret the results informally.

The informal nature of mathematical investigation as it is pursued in practice is in some respects better represented in recreational puzzle solving than in the standard mathematical exercises associated with a lecture course. As Andrew Hodges has said of Sudoku: 'Sudoku may not require long multiplication or division, but it is a very good puzzle that replicates the pattern of thinking required to solve quite complex logical problems in maths' [36]. The human interest in puzzles derives from the way in which they typically elude comprehensive systematisation, and engage the solver in thinking that does not follow routine paths. The emphasis is on the enjoyment derived from the moment-by-moment experience of organising information and understanding relationships, and the pleasure of meeting new challenges en route to finding a solution. The satisfaction that specific puzzles, or puzzles of a particular genre, give to the human solver tends to diminish as and when they are able to find mechanical approaches to their solution.

Human solving of Sudoku puzzles illustrates many of the key issues. Writing a computer program to solve a Sudoku puzzle (e.g. by exhaustive search) is a straightforward exercise. Such a program will not normally throw much light on the patterns of reasoning that the human solver must employ. In making an EM construal of the solving process, the focus is on tracing the solver's state of mind as they progress towards a solution rather than automating a mechanical process. This involves explicitly recording the solver's observation of the puzzle and maintaining those dependency relations that can realistically be apprehended at each step.

The notion of dependency in EM is illuminated by the way that it is applied in this context. There are several different senses in which the digit in a cell can be deemed to depend on the digits assigned to other cells. Provided that a Sudoku puzzle is well-posed, the digit that should occupy any given square is in fact determined from the outset of the solving process. But in considering Sudoku puzzle solution from a human

perspective, there is a crucial distinction between those assignments of digits that the solver can see "at once" to be consequences of elementary inferences, and those that become apparent only after many further steps of reasoning have been carried out.

The EM construal for Sudoku solving (see Figure 4) is similar in nature to the construal of bubblesort. It comprises routine observables to record the location of grid cells, the contents of all cells, whether the content is given in the puzzle or to be inferred etc, together with more sophisticated observables such as are associated with applying simple rules. The set of *plausible* digits that could presently be entered into a cell taking into account existing entries in the same row, column or region is one such observable.

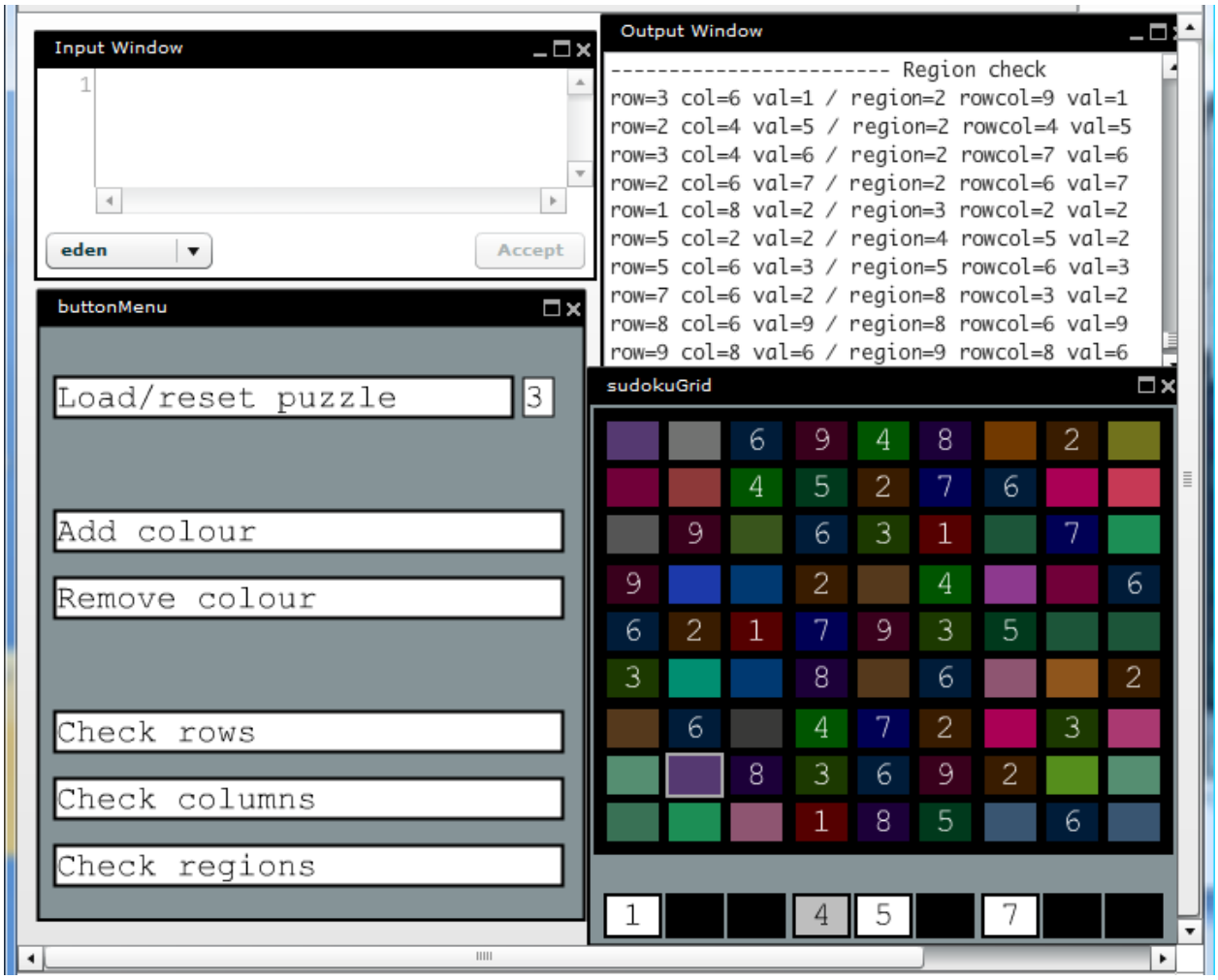

Figure 4: A screenshot from the EM model to support semi-automated Sudoku solving

Whereas it is relatively easy to identify the interactions and observables that are most relevant in the case of bubblesort, there is more freedom to consider new modes of interaction and observation in solving a Sudoku puzzle. This is because there is no universal set of standard rules to be followed in arriving at a solution, and human skill and speed of perception have a significant role to play. The incremental and open-ended way in which an EM construal is developed is well-suited to the solving activity in this respect. For instance, it is natural to extend the construal to allow the solver to record the fact that a digit that might plausibly be placed in a particular cell by the naive criterion described above is in fact ruled out by some other consideration. (Such a situation is illustrated in Figure 4, where the naive criterion indicates that 1, 4, 5 and 7 are plausible for the selected cell, but the choice of 4 is excluded by observing that this precludes placing a 4 in the region immediately above.)

Finding a Sudoku puzzle solution can not only be viewed of itself as a matter of disposition (viz. placing digits in the appropriate cells), but is influenced by disposition of a higher-level nature concerning the way in which the information garnered from observation and inference is organised and mediated to the solver. A relevant question in this context is how far automatic support for the human solver can simplify the solver's task without detracting from the pleasure of the solving process. The variant of the EM construal depicted in Figure 4 is a "Colour Sudoku" extension of a simpler construal in which all cells had a white background. In this variant, distinct colours are associated with the digits 1 to 9, and the background colour of a cell is a blend of the

colours of those digits that could plausibly be placed in it. This extension of the basic construal enables the solver to identify cells for which there is only one plausible digit by direct inspection and colour-matching.

The support that EM gives to a more holistic view of mathematics can be illustrated by considering an environment developed in association with a second-year module devoted to formal specification [34]. This environment is intended to illustrate how the automatic generation of a mathematical object from an axiomatic specification differs from, but can connected with, the ways in which mathematicians typically approach construction.

For the purpose of this illustration, groups with eight elements form a suitable source of examples. There are five such essentially different groups, all of which can be generated automatically from an axiomatic specification using a tool such as the Alloy Analyzer [37]. The five groups comprise three commutative groups, in which the product of a pair of elements is independent of their order (C8, C2×C4 and C2×C2×C2), and two non-commutative groups (the dihedral group D8 and the quaternions Q8). They illustrate a range of different ways in which groups can be manifest in concrete terms. For instance, they can be respectively identified with: addition modulo 8; multiplication of co-prime residues modulo 15; addition of subsets of {1,2,3} defined by symmetric difference; transformations that respect the symmetry of a square tile; and by the quaternions (a generalisation and extension of complex numbers that can be linked to transformations of objects in three dimensional space). The structure of these five groups is sufficiently rich to enable concepts such as subgroups, quotient groups and normal subgroups to be illustrated.

The students for whom the group display environment was developed had been introduced to basic group theory in a previous module. In such an introduction, aimed at computer science students, it is natural to emphasise those aspects of the mathematical theory that link most directly to the abstract foundations of computing. The use of Alloy to enumerate instances of groups with 8 elements by making inferences from the group axioms illuminates this formal stance. The limitations of such inference need to be appreciated, however, and a deeper appreciation of group theory has to come from a more informal intuitive engagement with specific examples and applications. Relevant activities might include: studying groups in the concrete forms in which they arise in number theory, geometry and linear algebra; analysing the structure of a group from its multiplication table; or displaying it as a finite automaton (as a group graph [38] or "Cayley diagram"). Students who do not play with concrete examples of groups of this nature typically fail to see the point of studying mathematical theory. They also fail to appreciate the essential character of theory, which is typically rooted in experimental and intuitive studies of concrete structures that precede axiomatisation.

Technological support for the intuitive exploration of groups presents quite a different challenge from computing support for automated formal analysis. It is necessary to enable a high level of exploration and experiment where the learner has greater autonomy and scope for making connections. The computer has historically served in the role of calculator, presenting the results to the user for off-line interpretation (albeit possibly now in visually much more impressive and informative ways than ever before), but it is the process of interpretation itself that now demands a dynamic blend of computer automated and manual human-directed interaction.

To fully enable this new kind of learning application, it is necessary to ensure that models of the learning domain, and artefacts to represent and manipulate conceptual objects within it, can be constructed by teachers and learners, not merely by specialist developers. This is an aspiration for EM; it is not yet realised in practice, but many fundamental elements are in place. A key unifying idea is that all interactions, whether they originate from the developer, the teacher or the pupil, are expressed as redefinitions of observables that correspond closely to meaningful observables in the learning domain. This is in contrast to the different ways in which the developer, the teacher and the pupil interact with conventional learning artefacts. Some of the qualities and potential can be illustrated by describing the development of the group display environment to date (see [16]).

The first step in constructing the group display environment had a serendipitous aspect. The Sudoku solving construal discussed previously supplied a 9-by-9 grid that was well suited to the presentation of an 8-by-8 group table, subject only to redefining the digits in cells and omitting superfluous observables that were relevant only to Sudoku solution. The association of a colour with each decimal digit was a feature of our Sudoku model that was retained, and subsequently simplified. Colour serves a useful function in concretising the abstract group elements. Though the concept of re-using and adapting a model in this way is within the scope of what a developer using object-oriented principles might consider, the nature of the underlying script that defines the EM construal of Sudoku here offers scope for a teacher to exploit re-use. Indeed, the Sudoku model can in principle be adapted on-the-fly by redefining the values of the appropriate observables appropriately. Conceptually, the dependencies to be discarded and introduced in this manner are easily grasped from knowledge of the intended application, and require only a modest amount of technical expertise.

The entry of an 8-by-8 group table into the Sudoku grid can be effected manually, but in order for the environment to serve its illustrative role, it was necessary to write a Unix shell script to translate the outputs of

the Alloy analyzer so that they could be presented in the group table. It was soon apparent that it was not straightforward to discriminate between one abstract group and another simply by inspecting the raw group table. It is possible to distinguish commutative from non-commutative groups by seeing whether the table is symmetric about the top-left bottom-right diagonal, but further analysis involves identifying the number of elements of a particular order. For instance, the group is the cyclic group C8 if it has an element of order 8, and is C2×C2×C2 if all its non-identity elements have order 2.

With a little experience and knowledge, this mechanical process of checking can lead to a reasonably speedy process of formal identification of instances of the five groups. It is not well-matched to the comprehension of group structure as a mathematician experiences it, however. For instance, a mathematician asked to construct the group table of C8 would develop a table in which row was a cyclic shift by one cell of the previous row. She might also identify the elements as 0,1,2,3,4,5,6,7 to reflect the addition of residues modulo 8. The EM interactive environment allows this process of re-organising the table and renaming its elements to be carried out with computer support by entering a small set of redefinitions (cf. the 5 definitions that are specified in the Input Window in Figure 5(a)) that can be recorded and replayed. Suitable reorderings and renamings that can be applied to the other four groups of size 8 are likewise specified and illustrated in the environment.

Further manipulation of the visual representations illuminates other basic concepts of group theory. To enable this, the original association of colours with digits in cells, as inherited from the Sudoku model, was overwritten by a simpler more explicit assignment of colours to digits, so that definitions to set the colour of one digit to that of another could be conveniently introduced. In this way, subgroups, quotient groups and normal subgroups can be identified with particular visual patterns that can arise through colouring and ordering group elements. For instance, displays (a) and (b) in Figure 5, derived from the original group table for D8 by making simple redefinitions of the element ordering, naming and colouring, expose {1,f} and {1,r,r2,r3} as subgroups, and reveal the latter as a normal subgroup for which the associated quotient is C2.

The openness of the modelling environment also allows additional components to be added by introducing a small file of definitions. For instance, Figure 5(c) depicts an extension whereby the learner can make the correspondence between elements of D8 and transformations of a square tile. The tile on the left depicts the original state of the tile, and that on the right its state when the transformation associated with the currently selected group element (as highlighted in Figure 5(b)) is applied.

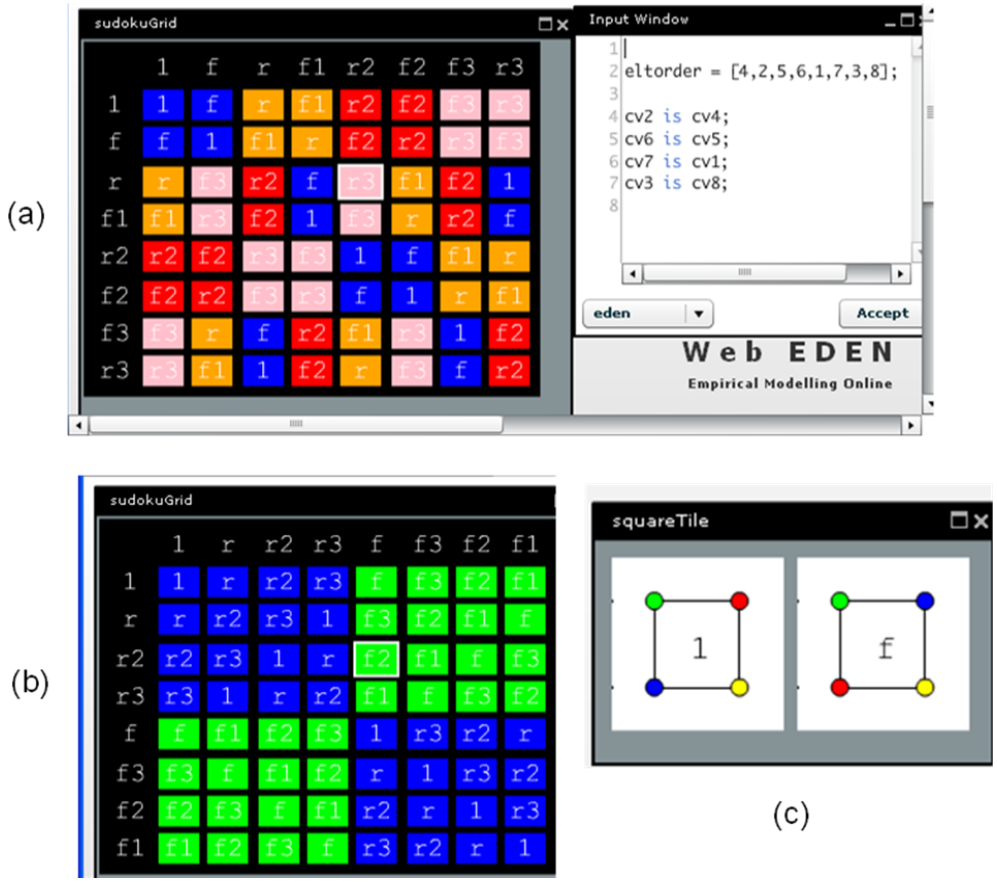

Figure 5: Displaying (a) a subgroup of D8 and the associated redefinitions of colours, (b) a normal subgroup of D8 and (c) the element f when interpreted as a transformation of a square

7. CONCLUDING DISCUSSION

When introducing a novel approach to computer science education, it is appropriate to evaluate and report the results. Despite the considerable body of work that has been outlined in this paper, objective evaluation of its impact has proved problematic. In the first instance, as the introductory orientation in section 1 makes clear, there are radical differences between constructivist computer science education as it is understood by Ben-Ari in [1] and as it has here been portrayed from an EM perspective. It would indeed be fair to say that the development of EM has been quite as much – if not more – concerned with challenging the accepted notion of computer science as with making learning artefacts to support mainstream computer science teaching. As documented by Antony Harfield in his doctoral thesis (see [23], p190-192), this difference in perspective between EM and computer science is appreciated by MEngCS students who are exposed to EM in a fourth year module. It is sufficiently great for it to be hard to understand the relationship between EM concepts and those of traditional computer science.

The experience of former doctoral students from the EM group is symptomatic of these difficulties. Yun Pui Yung was one of the first students to obtain a doctorate from the EM research group [39] and was a major contributor both to the thinking behind EM and to the development of the EDEN interpreter. In a personal communication in January 2009 [40], he writes:

“Occasionally I do think back and try to see how my experience in Warwick contributed to my current programming practice. One of the things that has been helpful is the concept of indivisible state transition. Of course I am not writing EDEN scripts at my work (I am mostly writing in JAVA and Perl), but I always do a mental check, if not writing a subroutine to do so, to make sure that the program state is consistent, and – when an exception condition occurs – that the state should also be resolved to a meaningful one. It definitively [sic] helped to me to write codes that have fewer bugs. I think this kind of "bridge" between "pure" EM and day-to-day programming practice may have mutual benefits. On the one hand, EM will gain more attention and on the other, practitioners can write better programs.”

Some of the most significant issues in comparing and contrasting an EM approach to CSE with a more conventional approach to CSE are exposed by revisiting the recommendations that Ben-Ari makes regarding constructivist CSE in [1].

The characteristic of EM that distinguishes it most forcefully from a more orthodox constructivist approach to CSE is its constructivist epistemological stance. For certain kinds of scientific mind, such a stance is anathema and seemingly gives licence to thinking that lacks the objectivity essential to many core computing applications. As Bruno Latour’s account of "the promises of constructivism" [41] vividly explains, there are good reasons to be sceptical of many proposals for a constructivist epistemology, and the needs of a science are not well-served by the extreme zeal with which some commentators have proclaimed the social construction of knowledge. Whereas responses such as that by Winograd and Flores [42] to too “rationalistic” an approach to computing are based on shifting the focus from formal languages to language as it is informally and contingently interpreted in social and cultural contexts, EM argues for a philosophical stance that is rooted not in language but in the immediate experience of personal interaction with artefacts [43]. In this respect, it is well-aligned to the 'radical empiricism' of William James [44,45] and to the philosopher of science David Gooding's notion of 'construal' [46]. It is also potentially well-placed to support Latour's vision for a notion of construction that is acceptable to science and to sociology [19,41]. Whilst many computer science students have been able to make important contributions to EM research without engaging with these philosophical issues, they alienate some.

Of course, the theoretical endorsement of distinguished thinkers counts for nothing without practical evidence that EM can generate viable models. Perhaps the most topical example of such a model is that developed by the author to teach relational database theory to undergraduates over the period 2001-3 (see [24]). As is well-known, there are discrepancies between the model of relational query evaluation used by SQL and Codd’s relational theory. By using EM principles, the author was able to develop an environment in which the interpretation of SQL queries in relational algebra could be realised either according to the appropriate strict mathematical conventions or taking those liberties with the semantics of sets and operations that are to be found in a standard SQL implementation. In realising this development, the scope for keeping in close touch with the current state of the environment and modifying its behaviour incrementally redefinition-by-redefinition proved to be highly significant. It might well have proved difficult to use a formal approach to specify the mathematically flawed evaluation model in SQL and at the same time enable the flexible switching between one evaluation strategy and another.

Ben-Ari’s analysis of constructivism in CSE [1] leads him to identify several important issues that impinge on teaching methods. Several of these have counterparts in the alternative constructivist framework of EM.

As remarked in the introduction, the emphasis that Ben-Ari places upon ensuring that the student has a viable model of the computer can be seen as related to the concern in EM for crafting the environment for interaction between agents prior to prescribing behaviours [13]. Such activity has been illustrated in the discussion of bubblesort. In this context, construals are playing the role of intuitive models of the (generalised) 'computer'. Whilst this seems to be in conflict with Ben-Ari's claim that such models are "doomed to be non-viable", it is important to bear in mind the significant distinction between 'a crafted environment in which reliable patterns of interaction between agents suitable to support specific-purpose behaviours have been identified' and 'a classical general-purpose computer equipped with a formal programming language'. The former is more typical of the kind of contextualised, but not typically formalised, mechanism that is identified by the engineer. And if the idea of a 'construal' to which EM appeals appears to lack credibility, its resemblance to the notion of construal introduced by Gooding in [46] should be borne in mind. Gooding after all invokes such a notion to explain how Faraday's experimental methods led him to develop the first electric motor.

Other parallels can be drawn between the conclusions concerning constructivist CSE to which Ben-Ari is led in [1] and the principles of EM. For instance:

- Ben-Ari discusses the need for icons to "undergo semiosis", so that "the user can *construct* a mental model of the object being represented". This brings to mind the manner in which the semantic relation between an EM construal and its referent is constructed (cf. the discussion of the filing cabinet model in section 2). A noteworthy difference in emphasis here is between signs that represent actions (such as 'paste'), and symbols that denote a complex entity (like a filing cabinet drawer).
- Ben-Ari stresses the need not to start with abstraction. This principle is clearly observed in EM, where the experiential – if not necessarily concrete – nature of the observables is at all times crucial. EM is clearly consonant with Piaget's principle that "abstraction follows assimilation". And whereas Ben-Ari criticises the premature introduction of object-like abstractions "to forget detail that you never knew or even imagined", the association of observables through dependency in EM is experientially mediated. The flexibility for modifying the associations between observables that modelling with dependencies affords brings to mind the feature of objects to which Ben-Ari alludes, viz. the role that modifying, extending and defining objects plays in creating abstractions. The relationship between a dependency and a functional program also echoes his endorsement of "models that can be explained in relatively high-level, hardware-free terms". Where a dependency cannot be formulated using the standard operators available in special-purpose notations in EDEN (cf. the notations whose use is illustrated in Figures 1, 2 and 3), there is provision in EDEN for specifying user-defined operators and such operators are to be interpreted as simple functional programs. They are, however, specified using a basic procedural programming notation. In this respect, current EM tools still rely at the most primitive level on knowledge of elementary constructs of conventional programming.
- Ben-Ari identifies bricolage and minimalism as features of a constructivist stance that have severe limitations where large-scale development is concerned. The aspiration in EM is to maintain a model throughout its development within an environment in which the semantic relationships between observables in the model and the external counterparts to which they refer can be readily made accessible in immediate experience. Practical experience of EM shows that apprehension of semantic relationships of this nature can be sustained for models with a few thousand definitions even with such an imperfect tool as EDEN (cf. the Sudoku model depicted in Figure 4, which comprises some 5000 observables). It is unsurprising that when we attempt to maintain a similar 'experiential' grasp of the relationship between a procedural program of several thousand lines and its intended behaviour, we typically flounder. Indeed, bricolage and minimalism are essential characteristics of EM, and their expressive range within the EM paradigm is yet to be decisively determined.

The research reported in this paper suggests that EM has promise as an alternative constructivist approach to computer science education. Wider adoption and support for tool and model development is essential – without this, it will be difficult to obtain sufficient objective evidence for the potential benefits we believe can be realised. A most encouraging characteristic of EM is the fact that the incremental construction of networks of dependencies is a process that can be recorded and re-enacted in precise detail. This feature has already been informally exploited in the reconstruction in 2007 by Harfield (see [23], p194-202) of a highly significant moment in a process of model-building carried out by an undergraduate student in November 2003. This highlights the scope for interesting and informative qualitative analysis through reconstruction of many kinds of activity that have been documented in the EM archive.

To date, the sole EM publication devoted to evaluation [47] is directed at EM education rather than conventional CSE. This reflects our primary interest in the future development of computer science itself. EM aspires to principles and tools that meet the needs of computing practice effectively in respect of both its formal and experiential aspects, and that have enough philosophical integrity to be an acceptable broader

basis for computing science. The issues that stem from the assimilation of computing into engineering [9] business [48] and the humanities [49] clearly point to the need for a wider conceptual framework for the academic discipline. A proposal for a wider curriculum for computing such as EM endorses was set out in connection with a workshop on *Thinking Through Computing* held at Warwick in November 2007 [50].

Though computer science is still a young and immature discipline, its indirect influence on the way in which human processes are conceptualised has been strong. The process by which computer science research in UK universities is currently monitored itself reflects this influence. There is a danger that this process, whilst giving due reward to the progress that is being made in computer science through specification, automation and optimisation, promotes the ossification of the discipline around a rationalistic, dualistic and linguistic philosophical stance. The future health of computer science as an academic discipline depends upon also recognising the potential for a science of computing that has stronger roots in our lived experience.

8. ACKNOWLEDGEMENTS

I am indebted to Steve Russ for his contributions to many of the ideas in this paper, to Ashley Ward, Chris Roe, Karl King and Antony Harfield for their help in developing and presenting the models referenced, and to Richard Myers for his implementation of Web Eden. I wish to thank Michael Jackson for introducing me to the quotation from Ferguson used in section 2. I am also much indebted to three anonymous referees for their insightful and stimulating comments on my original draft.

9. REFERENCES

[1] Ben Ari, M., Constructivism in Computer Science Education, *Jl. Of Computers in Mathematics and Science Teaching* **20**(1), 45-73, (2001).

[2] Wing, J., Computational Thinking, *CACM*, March, **49**(3), 33-34, (2000).

[3] Kramer, J., Is abstraction the key to computing? *Communications of the ACM*, **50**(4), 36 - 42, (2007).

[4] Dehnadi, S. and Bornat, R., The camel has two humps, In *Proc*. *Little PPIG*. Coventry, UK, (2006).

[5] Pacey, A., *Meaning in Technology*, The MIT Press, (2001).

[6] Brooks, F. P., No Silver Bullet - essence and accident in software Engineering, *Computer*, **20**(4):10–19, (1987).

[7] Harel, D., Statecharts: a visual formalism for complex systems, *Science of Computer Programming*, **8**, 323-364, (1987).

[8] Harel, D. and Marelly, R., *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, (2003).

[9] Jackson, M. What can we expect from program verification? *IEEE Computer*, **39**(10), 53–59, (2006).

[10] Ridley, M. J., Database Systems or Database Theory - or 'Why Don't You Teach Oracle', *Proc. LTSN-ICS Workshop on Teaching Learning and Assessment in Databases (TLAD).* Coventry, UK, (2003).

[11] Cantwell-Smith, B., The Foundations of Computing, in: M. Scheutz (Ed.), *Computationalism: New Directions*, MIT Press, (2002).

[12] Kallinikos, J., Farewell to Constructivism: Technology and Context-Embedded Action. In Avgerou et al. (eds): *The Social Study of Information and Communication Techonology*. Oxford University Press. 235-274, (2004).

[13] Boyatt, R. C., Beynon, W. M. and Russ, S. B., Rethinking Programming. In ITNG '06: *Proceedings of the Third International Conference on Information Technology: New Generations*, 149–154, (2006).

[14] Cantwell-Smith, B., Two Lessons of Logic, *Comput. Intell.* **3**, 214-218, (1987).

[15] The EM website http://www.dcs.warwick.ac.uk/modelling/, [Accessed 13/06/09].

[16] The Web EDEN webpage, with links to the EM models discussed in this paper http://www.warwick.ac.uk/go/webeden, [Accessed 13/06/09].

[17] King, K., *Uncovering Empirical Modelling*, MSc Thesis, Department of Computer Science, University of Warwick, UK, January, (2007).

[18] Beynon, W.M., Computing technology for learning — in need of a radical new conception. *Journal of Educational Technology and Society*, **10**(1):94–106, (2007).

[19] Beynon, W.M. and Harfield, A., Lifelong Learning, Empirical Modelling and the Promises of Constructivism. *Journal of Computers*, **2**(3), 43–55, (2007).

[20] Harfield, A., Beynon, M., and Myers, R., Web Eden and Moodle: an Empirical Modelling approach to web-based education. In *Proceedings of the Eighth IASTED International conference on Web-Based Education*, March 16-18, 272-278, (2009).

[21] Beynon, W.M. and Harfield, A., Empirical Modelling in Support of Constructionist Learning: a Case Study from Relational Database Theory. In: *Proc 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05)* Kaohsiung, Taiwan, July, 396-8, (2005).

[22] Roe, C., *Computers for Learning: An Empirical Modelling perspective*. PhD thesis, Department of Computer Science, University of Warwick, UK (November 2003). http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/phd/croe/. [Accessed 28/05/09].

[23] Harfield, A. *Empirical Modelling as a new paradigm for educational technology*. PhD thesis, Department of Computer Science, University of Warwick, UK (January 2008). http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/phd/ant/. [Accessed 28/05/09].

[24] Beynon, W.M., Bhalerao, A.H., Roe, C. and Ward, A., A computer-based environment for the study of relational query languages. In: *Proc. of the Teaching, Learning and Assessment in Databases Workshop*, Coventry, UK, July, 104-108, (2003).

[25] Beynon, W. M., Rungrattanaubol, J. and Sinclair, J., Formal Specification from an Observation-Oriented Perspective. *Journal of Universal Computer Science*, **6** (4), 407- 421, (2000).

[26] The EM archive http://www2.warwick.ac.uk/fac/sci/dcs/research/em/projects/index/ [Accessed 13/06/09].

[27] Ferguson, E., *Engineering in the Mind's Eye*, The MIT Press, (1992).

[28] Mason, J., Representing representing: Notes Following the Conference. In: Janvier, C. (Ed.), *Problems of Representation in Teaching and Learning Mathematics*, Hillsdale, NJ: Lawrence Erlbaum, 207-214, (1987).

[29] Turski, W. M. and Maibaum, T. S. E., *The Specification of Computer Programs*. Addison-Wesley Publishing Company, (1987).

[30] Baker, J. E. and Sugden, S. J., Spreadsheets in Education – the First 25 Years, e-Journal: *Spreadsheets in Education,* **1**(1), 18-43, July, (2003).

[31] Nardi, B., *A Small Matter of Programming*. MIT Press, (1993).

[32] Horrocks, I., *Constructing the User Interface with Statecharts*. Addison-Wesley, (1999).

[33] Jullien, F. *Praise of Blandness: Proceeding from Chinese Thought and Aesthetics*, trans. By Paula M. Varsano, MIT Press, (2004).

[34] Beynon, W. M., and Klein, R. R., Métis meets Empirical Modelling: from ancient wisdom to emerging technology. In: Cunningham, P. and M., (eds), *IST-Africa 2006 Conference Proceedings,* IIMC International Information Management Corporation, (2006).

[35] Byers, W., *How Mathematicians Think: Using Ambiguity, Contradiction, and Paradox to Create Mathematics*, Princeton University Press, (2007).

[36] Hodges, A., - as reported by John Crace in the Guardian, Tuesday 13 November 2007. See url: http://www.guardian.co.uk/education/2007/nov/13/schools.uk [Accessed 17/06/09].

[37] Jackson, D., *Software Abstractions: Logic, Language and Analysis*. MIT Press, (2006).

[38] Grossman, I., and Magnus, W. *Groups and their graphs*. The Mathematical Association of America, (1964).

[39] Yung, Y.P., *Definitive* Programming: A Paradigm for Exploratory Programming. PhD thesis, Department of Computer Science, University of Warwick, UK (October 1992).

[40] Yung, Y.P., personal communication, 5th January 2009.

[41] Latour, B., The Promises of Constructivism, In Ihde, D. (ed.) *Chasing Technoscience: Matrix of Materiality*, Indiana University Press, (2003).

[42] Winograd, T., and Flores, F., *Understanding Computers and Cognition*, Addison-Wesley, (1987).

[43] Beynon, W.M. and Russ, W., Experimenting with Computing. *Journal of Applied Logic,* **6**, 476-489, (2008).

[44] James, W., *Essays in Radical Empiricism*, Longmans Green, (1912).

[45] Beynon, W.M., Radical Empiricism, Empirical Modelling and the nature of knowing. In: Itiel Dror (ed.) *Cognitive Technologies and the Pragmatics of Cognition: Special Issue of Pragmatics and Cognition*, **13**, 615-646, Dec, (2005).

[46] Gooding, D., *Experiment and the Making of Meaning: Human Agency in Scientific Observation and Experiment*. Kluwer Academic Publishers, (1990).

[47] Boyatt, R., Harfield, A., and Beynon, W.M., Learning about and through Empirical Modelling. In: Proc 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006) Kerkrade, The Netherlands, July, 662-666, (2006).

[48] Hirschheim, R., Klein, H.K. and Lyytinen, K. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*, Cambridge University Press, (1995).

[49] McCarty, W., *Humanities Computing*, Palgrave Macmillan, (2005).

[50] Beynon W.M., and Russ, S., An EM Perspective on Computing. Presented at the “Thinking Through Computing” workshop, University of Warwick, November 2007. http://www2.warwick.ac.uk/fac/sci/dcs/research/em/thinkcomp07/ [Accessed 13/06/09].