Research Report 346

Explanatory Models for Open-Ended Human-Computer Interaction

M Beynon, J Rungrattanaubol, P-H Sun, A Wright

RR346

Design principles to support human-computer interaction outside a closed-world framework are considered. These are based on the construction of explanatory models using empirical Modelling tools and methods devised at Warwick. The approach can be applied to interaction that is associated with evolving understanding and the acquistion of skills on the part of the user. This is illustrated with reference to a study in mathematical investigation and visualisation, and the construction of a family of interactive models that can be used to teach the **heapsort** sorting algorithm.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

July 1998

Explanatory Models for Open-Ended Human-Computer Interaction

Meurig Beynon, Jaratsri Rungrattanaubol, Pi-Hwa Sun, Amanda Wright
Department of Computer Science, University of Warwick, Coventry CV4 7AL

Abstract

Design principles to support human-computer interaction outside a closed-world framework are considered. These are based on the construction of explanatory models using Empirical Modelling tools and methods devised at Warwick. The approach can be applied to interaction that is associated with evolving understanding and the acquisition of skills on the part of the user. This is illustrated with reference to a study in mathematical investigation and visualisation, and the construction of a family of interactive models that can be used to teach the ***heapsort*** *sorting algorithm.*

Keywords: conceptual models for HCI, cognitive frameworks for HCI, designing HCI, computer-assisted learning

Introduction

Trends in human-computer interaction suggest that, in the future, interaction with computer-based technology will come to resemble our everyday interaction with the world. Research in Virtual Reality has demonstrated the benefits of developing interfaces that mimic realistic interaction within closely circumscribed domains (such as brain surgery). Despite such advances, it is facile to imagine that new technologies alone can make our engagement with computer models similar to our engagement with the world in its fundamental character. Everyday experience of our environment is characterised by an openness to interact, and an ignorance of the consequences of our interaction, that has no counterpart in interaction with closed-world computer models.

For good reason, the predominant emphasis in computer-based modelling has so far been on constructing models that reflect our comprehensive understanding of a domain, and attaching interfaces that diminish the demands upon the user's intelligence. In such a context, domain knowledge informs the construction of the computer model during the design phase, and the ways in which users can subsequently process and supplement this knowledge is preconceived by the designer (see Figure 1). The design of the computer model is then *user-centred* [12]; it is guided by how domain knowledge is to be processed so as to best meet specific user needs. This typically means that users are insulated from the experimental activities that informed the relevant domain knowledge; the computer model merely automates a particular set of ways of processing this knowledge for a specific class of users, and is optimised for this sole function. The user interaction takes place within a closed world bounded by the understanding of the domain, and of the expected uses to be made of domain knowledge, that is accessible to the designer.

This paper describes a paradigm for developing interfaces where the emphasis is not merely upon interaction of the closed-world variety. To achieve this end, both the designer and the user operate within a common computer-based environment whose state metaphorically represents the experiential context from which the domain knowledge is derived (the "explanatory model" in Figure 2). A spreadsheet is one example of an environment of this nature: the cells in the spreadsheet represent the values of particular external observables, and the changes to the values of cells that occur in the process of updating the spreadsheet faithfully reflect the patterns of change to families of observables in the experience of the user. The most efficient way to process the spreadsheet data in a specific way is to develop special-purpose programs that exploit the data dependencies rather than record them explicitly. The advantage of storing the data dependencies is that users are empowered to devise their own extensions and experiments. For instance, a user can revise the way in which tax liability depends on income to reflect a change in tax law.

The paper is in two sections. The first describes how human-computer interaction for open development can be supported using tools and methods from our Empirical Modelling framework. The second outlines and illustrates their application to computer-based environments for pedagogical use (cf [3]).

2. Human-Computer Interaction from the Empirical Modelling (EM) perspective

2.1. The Nature of EM Models

The objective of the Empirical Modelling (EM) process can be seen as the construction of a computer environment (hereafter EM model) that metaphorically represents a state of the external world. The construction of such a model is *situated*, in the sense that some independent context informs its development. Before introducing the principles that are used to represent state within an EM model, it is useful to consider a specific example.

By way of illustration, the EM model may be a line drawing of a room, showing the current positions of the furniture. The room may be a specific room, such as the one in which the modeller actually carries out the modelling process. It may exist only in the modeller's imagination. The most significant aspect of the room as a referent is that the modeller can treat the room as an entity that can be experienced. That is to say the room has an identity and integrity that persists despite the fact that the room as currently apprehended is subject to change. The modeller can attach meaning to a concept such as: this is the *same* room, but in a *different* state. An EM model is to be regarded as a qualitatively similar representation of a particular room. A different state of the same line drawing depicts the same room in a different state. (For an account of the significance of treating "perception of identity" as a primitive element of experience, see James [10].)

The concept of the *state* of the room is subtle. What is deemed to be a change of state depends upon what characterises identity and integrity. It may be that the modeller's interest is in exploring possible room layouts, but each layout will be interpreted as different states of one and the same room. An architect may revise the shape of a room several times, and regard each revision as a new state of the same room. The idea of the referent as an element of experience embodies the possibility that what can be remarked by way of characteristics of a particular state is open-ended. For instance, to attach to a room layout the observation that in this room layout the door is obstructed by the table is not to change the state of the room, but to acknowledge an additional feature of its current state. In an EM model, there is a similar distinction between incorporating new features into the current state, and changing the state.

Although an EM model may be seen as sharing some of the characteristics of a Virtual Reality environment, realistic representation is a possible rather than a significant concern. A line drawing of the room layout is of limited use as a model for an interior designer, who would prefer a photorealistic image of the room. In contrast, when specifying the current state of an electrical device, it may be appropriate to construct a circuit diagram which is annotated with present values of significant voltages and currents. Inspecting the circuit diagram in no way resembles inspecting the device, but there is nonetheless a direct relationship between features of the circuit diagram and their annotations, and components of the device and their current status. The openness of the EM process is indicated by the fact that a single EM model can (in principle) incorporate such diverse representations of the current state of room as a photorealistic image, a two-dimensional room layout, and a circuit diagram with dynamic annotations to represent the status of the electrical wiring.

The construction of EM models for a wide variety of systems has been extensively discussed in previous papers [2,3,4,5,6]. The EM process combines two fundamental principles: observation-and-agent-oriented analysis and definitive (definition-based) representation of state. The remainder of this section explains how the development of the EM model is associated with experimenting on and exploring the referent, and briefly outlines the principles that are used in constructing an explicit computer model. Empirical knowledge has a quite different character from theoretical knowledge. It is always open to subsequent extension, refinement and revision in the light of future experience. Since EM is concerned with the empirical processes through which knowledge and understanding are acquired, an EM model is in itself essentially incomplete, and must be complemented by interaction. As section 3 illustrates, this means that EM models can be exploited as interactive environments for learning and communication.

2.2. Interaction and the Development of EM Models

Constructing an EM model for a referent is associated with learning through interaction. This process is quite different in character from being taught through activities that focus on the use of language. An appropriate perspective to consider is that of a child's first encounter with the world, or our own interaction with an unfamiliar computer-based adventure game. This interaction initially has a tentative, exploratory character, and progresses by degrees to the point where assurance has been gained. Even the potential referents - those elements of the situation with which it is possible to become familiar - are uncertain at the earliest stages. A fundamental problem in our interaction is to determine how we can act on our environment, and to what extent we can choose specific actions and control their effects. We may first become aware of potential actions inadvertently, and initially have difficulty in distinguishing between exceptional and routine interactions. Only at a relatively advanced stage are we able to identify relationships between features of our situation and particular privileges to take action.

Our exploratory interaction is further complicated by the presence of other state-changing agents. Like the experimental scientist, we have to identify stable contexts for interaction with our environment, develop instruments to enhance our modes of observation and interaction, and acquire knowledge and skills of appropriate experimental procedures. The counterpoint between these activities is well-illustrated in the history of instrument design [9]. For example, in the early 19th century, scientists took time to recognise that changes to the recorded measurements of the earth's magnetic field over a period of years could not be attributed to new instrumentation alone. Instrument development of this nature has contemporary relevance for computer-based technology.

The EM process revolves around three fundamental concepts: observables, dependencies and agents. The term *observable* is used to refer to elements of a situation that have identity and integrity. For instance, observables in a room (as captured by an annotated room layout) might include the walls, the corners of the room, the furniture in the room, and the features of objects such as doors, drawers and switches that can be open or closed and on or off. They might also include voltages and currents in electrical devices in the room, and conceptual aspects of the state, such as whether the door is currently obstructed, or whether the room requires cleaning. A *dependency* is a relationship amongst observables that expresses expectations about how the values of observables are indivisibly linked in change. When the light switch is pressed on, there is a current in the circuit and the light comes on at one and the same time. When the table is moved, the door is no longer obstructed. When a new piece of furniture is introduced the free floorspace in the room is reduced. An *agent* is an observable (generally identified with a collection of more primitive observables) that is deemed to be responsible for changes of state within the situation. Examples include: a user of the room who may move furniture, open or shut the door, or put the light on; a light switch with a time-delay that switches off the light after a period of time; a sensor that puts the light on when movement is detected in the room.

The identification of observables, dependencies and agents is an empirical process that leads to a provisional explanation for what is being observed. This process involves apprehending the integrity of the referent, and distinguishing observation of the referent from ephemeral observation, of which it is impossible to say "this is further observation of the same thing" (cf. distinguishing "more accurate measurements of the - supposedly time-invariant - earth's magnetic field" from "changes to the earth's magnetic field over time"). The product of the EM process is a causal account for the observed response to interaction and autonomous behaviour of the referent. This account can be regarded as a provisional and generally incomplete framework within which to interpret future experience of the referent, as developed on the basis of previous observation and experiment. Such a causal account includes information about what agents are potentially involved in the referent, and which are present in any particular state. It also indicates which observables are associated with which agents, which observables are deemed to affect the state of an agent, and which observables are conditionally under the control of each agent.

2.3. Empirical Modelling Methods and Tools

In the course of the Empirical Modelling project, several software tools have been developed for constructing models. These will be briefly outlined in this section. For further details, the interested reader may consult the EM website: http://www.dcs.warwick.ac.uk/pub/research/modelling, and many previous publications.

An EM model is based upon a family of observables, dependencies and agents. Such a model is presented to the modeller in a particular state, and the possible changes of state are in principle constrained only by knowledge that is at any time open to revision in the light of new experience. For this reason, it is generally convenient to focus on establishing an explicit state representation, but allow the more subtle aspects of agency to be implicit in the mode of interaction with the model. For instance, in studying room layout, it may well be appropriate to simulate the actions of a room user, but the precise nature of the simulation that is required in a particular situation is uncertain, and cannot be preprogrammed. We may want to determine whether it is possible to reach the light switch from a chair, or whether one piece of furniture can be moved around another. Explicit knowledge of the capabilities and characteristics of agents is useful in this context, but the way in which these are exercised is specific to the modelling task.

In an EM model, observables and dependencies are represented by a family of definitions ("a definitive script"). The current values of observables and dependencies represent the current state of the referent. A typical interaction with the model involves redefining a variable, or introducing a new variable and its definition. The interpretation of such interactions is situation-dependent. For example: the redefinition of a variable may represent a change of state in the referent or a revision of the model (such as the correction of an observed measurement). Introducing new definitions may reflect a change in the state of the referent (such as the advent of a new agent), or a refinement in the observation of the current state.

The special-purpose notation LSD has been developed to specify the characteristics of agents in respect of interaction within their environment. The observables associated with an agent are classified as *states*, the observables are deemed to influence the agent as *oracles*, and observables that are conditionally under its control as *handles*. The dependencies between observables that characterise the interaction of the agent with its environment are *derivates*, and its privileges to act are expressed as guarded sequences of redefinitions of handles and agent invocations or dismissals. A causal account of the referent can be expressed in LSD, and can be used to guide the construction and evolution of the EM model. The modeller potentially acts in the capacity of a superagent in interaction with the definitive script that defines the current state, and the way in which this agency is exercised can be interpreted as determining the laws that govern the behaviour of the referent. The modeller may also be concerned with the agency of potential users. The account of the how interaction informs a human agent's view of an unfamiliar world in the previous section can be reinterpreted as a process of classifying observables in LSD terms, both in respect of the agent itself and in respect of other agents. Together with the definitive representation of the current state, this supplies the nucleus of the open-ended and ever evolving "explanatory model" (see Figure 2).

The tkeden interpreter is the principal modelling tool that has so far been developed. Tkeden supports definitive scripts for line drawing and window layout and allows the user to establish dependency relationships between scalars, list and strings using built-in and user-defined functions. There is as yet no explicit automatic means to introduce LSD agents into the EM model, but these can be represented by actions in the form of triggered procedures that execute in response to changes in variables. This makes it possible to introduce autonomous agents (such as a time-delay light switch), and to provide the modeller with an interface for invoking agent actions as and when appropriate. A distributed variant of the tkeden interpreter has recently been developed by Pi-Hwa Sun. This allows several modellers to cooperate through communicating definitions and actions within a client-server configuration of tkeden interpreters.

3. Interaction and Understanding

EM models can be exploited in two ways. In domains that are sufficiently well-understood, it is possible to attach an interface to an EM model to derive computer programs with conventional functionality [2]. This mode of application follows a traditional pattern of engineering, where empirical insight shapes the context for the design of an engineering product to fulfil a specific role. Experimentation, ingenuity and intelligence inform the design of a device, but are hidden from the user. The design process typically involves many decisions that could have led to variations in form or function, but these are obscured through optimisation for the specific design and choice of role. Alternatively, an EM model can be used in conjunction with interfaces that invite the user to engage with the empirical process behind its construction. For instance, interaction with the model can be directed at imparting relevant domain knowledge to the user.

Though EM has been applied to the construction of computer models associated with engineering products (cf. the Vehicle Cruise Controller in [6]), these do not do justice to the empirical knowledge that informs their design in practice. The principles of developing educational environments from EM models are best illustrated with reference to models that have been developed for a specific pedagogical purpose.

Two examples to illustrate the application of EM models for teaching will be described. The first is based on a study in mathematical visualisation that has been discussed in detail in a previous paper [4]. The original model was developed by the principal author in 1991. The account below draws on the work and experience of Rungrattanaubol, who has extended the model in various ways with only a limited degree of consultation with its original developer. The second illustration concerns the construction of an EM model that exposes the principles of heapsort [5]. This model gives a simple visual representation of a heap that can be used interactively to demonstrate the heap concept, to allow students to experiment with heap structures and operations upon them, and to support execution of the heapsort algorithm, both user-driven and automatic. The construction of this model, developed by the principal author with the assistance of Wright, is the first instance of collaborative use of EM methods in a distributed environment. The construction and extension of both EM models will be used to illustrate fundamental principles concerning the relationship between modes of interaction and the acquisition and communication of insight. The most significant feature of both these models (cf. Figure 2) is that they are always open to extension by user and designer alike, and, through the adjunction of agents and interfaces, can support a plethora of HCI models.

Certain general principles concerning interaction and learning are common to both examples. Both examples demonstrate how understanding originates with developing expectations about the response to interaction. This can take the form of local knowledge about the effect of a single action in a particular state (as in the identification of a dependency between observables), but can also involve developing insight into invariant relationships between observables with respect to a family of possible sequences of interactions. Emergence is relevant here, in that precise knowledge of what each individual interaction achieves does not account for their corporate effect.

Knowledge of how to achieve particular relationships between observables is complementary to theoretical knowledge about universal relationships between observables. An EM model can incorporate control observables (such as are represented for instance by the status of menu buttons on an interface) that make the association between situation and potential action apparent to the user (cf. "painting by numbers"). If visual cues of this kind are absent from the model, the identification of control observables makes demands on the user, and typically involves learning through observation and experiment. This illustrates a general principle: intelligent action on the part of the user is only necessary where there is a degree of autonomy, and effective learning presumes greater automony. Similar considerations govern the extent to which interpretation of the current situation is explicitly communicated to the user through the interface: the user who can read a textual message need not be aware of how to derive its content from observation of the situation. This helps to clarify the status of explanation, and the extent to which documentation of state can inhibit rather than promote a deeper understanding.

3.1. Modelling combinatorial properties associated with configurations of lines

In its original form, the *lines* model consisted of a script that comprised some 400 definitions (see Figure 3). The principal purpose of constructing this script was to allow interactive exploration of the relationship between a configuration of lines (indexed by the numbers 1 to 4 in Figure 3), and the combinatorial patterns that can be derived from such a configuration by interpreting the intersection of a pair of lines as the transposition of a pair of indices. (For more details of the significance of this study, see [4].)

The process of constructing the model required an intricate and systematic analysis of the dependencies between features in the various components of the visual display. The variables introduced into the script in this process can be interpreted with reference to these features. Consider the following brief extract from the script, for instance:

```
x12 = 1+Z123+Z124
...
Z123 = if r13 < r12 then 1 else 0
Z124 = if r14 < r12 then 1 else 0
...
r12 = a12/b12
r13 = (a12+a23)/(b12+b23)
r14 = (a12+a23+a34)/(b12+b23+b34)
```

In this extract, the variables a12, a23, a34 and b12, b23, b34 respectively determine the relative distances between the LH and RH endpoints of the lines 1 through 4; the variables r12, r13, r14, r24, ..., r34 correspond to the x-coordinates of the points of intersection between lines 1 and 2, 1 and 3 etc; the variables Z123, Z124 etc are such that Z123 has value 1 or 0 according to whether lines 1 and 3 intersect to the left or to the right of lines 1 and 2, and the variable x12 gives the x-coordinate of the node that represents the intersection of lines 1 and 2 in the partially-ordered set at the top right in Figure 3.

The typical use of the model, which was the primary motivation for its construction, involves re-assigning values to a12, a23, a34, b12, b23, b34 and observing the effect upon the combinatorial pattern of intersections. An interface can be added to the model in such a way that precisely this functionality is available to the user. Such an interface makes it simpler to use the model to serve an abstract mathematical function, and investigate theoretical or emergent properties (e.g. exploring which products of transpositions can arise from a configuration), but gives no insight into how the model has been constructed and could be modified to serve a new function. In this respect, the interface protects the user from deeper knowledge, and from potential sources of insight and avenues for experiment.

It is instructive to note that the developer is also subject to a similar loss of insight into the inner workings of the model. The unusual qualities of the EM model are illustrated by the fact that this insight can be reconstructed through direct interaction with the model similar in character to the interaction with a real-world referent discussed in section 2. Rungrattanaubol was able to show this by performing experiments on the model, adding simple scripts to reveal the values of internal variables, interrogating the dependency relations, changing the values of variables by direct assignment and observing their effects, and adding simple interfaces to allow convenient monitoring of responses to changes in key parameters. In this process, she was able to deduce the semantics of variables, and so interpret the detailed mechanisms used to maintain dependency. In reporting on this process, she remarks upon the way in which exploration of the model leads to understanding that is first manifest as skill in manipulation and observation of the model, and can only later be articulated. This indicates how skill in interaction with EM models can represent forms of intuition about structure that precede precise identification.

The documentation and interpretation of a large definitive script presents rather different challenges from a conventional program. There is no natural order for the definitions, for instance. Indeed, different ways of grouping definitions together suit alternative viewpoints on the structure of the visualisation, to reflect, for example: their nature (pertaining to window layout, line drawing or scalar values internal to the model); their semantic interrelationship and dependency (cf. the above extract); their association with particular entities (e.g. details of the visualisation comprising lines, points, scalars and labels associated with a particular line of the configuration). Conventional annotation of the script is particularly problematic, as a comment generally takes the form of a statement about a subset of definitions from the script, and this presumes a particular ordering.

One form of documentation that has been investigated involves adding new observables to the model that serve as interpretations of existing observables (cf. Figure 4). For instance, as remarked above, *Z123 has value 1 or 0 according to whether lines 1 and 3 intersect to the left or to the right of lines 1 and 2*; by consulting this interpretation and the current value of Z123, it is possible to add an explicit dynamic annotation to the visualisation, to the effect that (when Z123=1, for instance)
lines 1 and 3 intersect to the left of lines 1 and 2.
The introduction of such annotations demonstrates the potential for presenting information about the situation pertaining in the model in ways that presume little intelligence on the part of the user. A literate observer can communicate this knowledge without needing to interpret the configuration. A fuller understanding of the model is gained by studying the dependency relations in the script. Even then, interpreting a dependency relation resembles being told the outcome of an experiment, and it can be argued that the most satisfactory understanding is obtained through personal experience of experimenting with the model. This discussion illustrates that explanation within EM models can operate at many levels of abstraction, and can be oriented towards different modes of use and kinds of user.

That empirical investigation of an EM model can lead to comprehension is indicated by the fact that Rungrattanaubol was able to carry out several independent extensions of the model of Figure 3. These included the development of new interfaces, generalisations of the model to deal with more lines, the addition of dynamic annotations, and the modification of the mechanism by which the number of minimal triangular regions in the configuration was computed. This indicates that the EM model can serve a novel role in communication, embodying insight in an implicit form suitable for recovery through interaction.

3.2. An EM Model for Heapsort

Unlike the *lines* model, which is aimed at exploring mathematical relationships, the EM *heapsort* model is concerned with exposing the empirical knowledge contributing to design of an algorithm. The model itself was constructed through on-line collaboration between the principal author and Wright, with a number of objectives that included teaching EM principles, creating an environment for introducing heapsort, and exploring the benefits of distributed use of the client-server variant of tkeden.

Details of the construction of the heapsort model and its potential applications are given in [5]. The following discussion will focus upon the issues most relevant to human-computer interaction, with particular emphasis upon how the model can assist communication and learning. Familiarity with the heapsort algorithm [1] will be helpful in understanding the discussion, but is not essential.

The development of an EM model for heapsort involves identifying the principal features of the heap data structure that are significant in the operation of heapsort. To appreciate the heapsort algorithm, it is necessary to understand how the elements of an array can be viewed as a binary tree, subject to regarding the elements indexed by 2i and 2i+1 as the children of the element with index i (see Figure 5). This tree is called a *heap* if the heap condition holds at each node, i.e. the value associated with each node is larger than the values at each of its children. More generally, the array elements with indices in the interval [*first*, *last*] form a heap if the heap condition is valid at all nodes within the segment of the tree derived by discounting

the nodes with indices less than *first* or greater than *last*. Figure 6 depicts a heap on the elements in the range 1 to 6 such as is generated by the heapsort algorithm as the elements are being output in sorted order.

The core of the heapsort model is a script that includes observables to represent an array of scalar values, its associated tree structure, the indices *first* and *last*, the order relations on the edges of the tree, the heap conditions at the nodes, together with the visual representations (as in Figure 6) for the nodes and edges. This script was developed in its earliest stages by developing a model for a 7 element heap, and restricting *first* and *last* to the values 1 and 7 respectively. This has the advantage of simplifying the heap condition criterion, as is typically done when first examining the heap concept. In its most basic form, the script simply represents the way in which the values at the nodes of the tree, the order relations on the edges and the heap conditions at the nodes depend upon the values in the array. At this stage, the user can redefine the values of the array elements, and see how this influences the significant features of the tree.

The next phase of the model construction is to add the visualisation of the tree and the array. This is done in such a way that the colour of edges and nodes reflect the order relations and heap conditions. Introducing this visual dependency has a similar effect to the annotation of the *lines* model discussed above: the tree is a heap if all the edges are blue, and a user familiar with this convention needs no deeper understanding of the model to be able to recognise a heap. The user nonetheless needs intelligence to be able to reorder the elements of a given array so as to obtain a heap. Since heapsort proceeds by exchanging elements, it is appropriate to introducing an agent to exchange the values associated with nodes with a given pair of indices. A suitable exercise for the user in this context involves establishing the heap condition by repeatedly exchanging values at nodes that are connnected by an edge.

At this point an intelligent user may recognise that the most effective strategy in establishing a heap is to exchange the value at a node with the value at whichever of its children has the greater value. The index of the child of a node with which the greater value is associated can be introduced as an additional observable. The model can now be extended by attaching agents to nodes that monitor the heap condition and, if the heap condition is violated, are privileged to exchange the value at the node with the value at the appropriate child. The characteristics of the interactive environment that results from introducing these agents depend crucially on how they are programmed to act. If an agent autonomously carries out the exchange as soon as the heap condition is violated, the process of heap construction is automated, and by default will be presented to the user as an indivisible updating operation. Under another control regime, the agents can act one by one in a non-deterministic fashion as prompted by the user (e.g. via a mouse click in the window that displays the tree). This allows the user to inspect the process of establishing the heap step-by-step, but need involve no action other than giving the prompt. A more demanding interface for the user requires that the next agent to act be invoked explicitly (e.g. by clicking at the associated node of the tree). In this case, the user has to select a node at which the heap condition is violated at each stage. The user's understanding can be tested in a different manner by eliminating the visual cues that distinguish the nodes at which the heap condition is violated.

In detailing these various interfaces, the intention is to show how the same EM model can readily be adapted to serve a variety of roles in the learning process, subject to modifying the protocols for internal and external interaction in simple ways. A variety of interactive models of this nature can be invoked interchangeably within the complete heapsort model by introducing short files to make simple modifications to the external interface and the internal agency (cf. Figure 2).

Similar principles apply to the transformations that are applied in order to elaborate the heapsort model. This involves allowing the variables *first* and *last* to take on different values, and modifying the definitions of the heap conditions, the display conventions for nodes and the index of the child with the greater value accordingly. Simple files that redefine these variables can be introduced to switch between the general and the restricted models of a heap. Either model can be complemented by agents with specific roles associated with primitive operations in the heapsorting process. A whole family of models can be derived by an

appropriate selection of heap model and internal agents. This includes a model in which the user can carry out the heapsorting process by explicitly specifying the steps in the algorithm, and another in which the heapsort process is carried out automatically in response to prompts from the user. Through attaching different interfaces in this way, the degree of assistance that is offered and explanation that is given to the user can be freely altered to suit specific educational purposes.

Collaborative development of the heapsort model was interesting as an exercise in communication. The earliest stages of the development process involved collaboration on a single machine. This did not offer autonomy to the two participants, and inhibited interaction with the model. For the experienced user, in the role of a teacher, interaction with the model is helpful as a way of assessing the current status of the model, and confirming the viability of a particular strategy for further development. For the inexperienced user, in the role of a student, interaction is useful as a way of checking comprehension and resolving obscure or uncertain issues. Private interactions with the model are useful and desirable for both parties, not least because problematic public interactions can undermine confidence in the competence of either collaborator or the integrity of the model.

The distributed modelling phases of the collaboration were far more effective in demonstrating the merits of EM models. Useful activities that could be distributed included: replication of definitions from a template, refinement of definitions from an outline, testing of the model, and the extraction of submodels. Exchanging a small file of definitions served to restore models to a consistent state, to communicate problematic scenarios, and to introduce extensions to the model. In the process of revising the heap condition to allow *first* and *last* indices to be changed, the logical definition was at first incorrectly specified. This emerged after interaction with the model, and could be corrected with direct reference to the empirical context by which it was informed. Interaction of this nature is particularly useful in revealing the fallibility of even the experienced developer, and in exposing the empirical processes behind the conception of logical expressions.

Conclusion

The research described in this paper has demonstrated the potential for applying Empirical Modelling to the development of computer models that support richer forms of exploration, interaction and communication. Future research will address the role that LSD accounts of agency can play in clarifying the issues that surround the understanding and skills of the user and the responses of the computer model. More practical experience and evaluation of the methods is also required. In due course, these should include studies involving interaction with models with more direct real-world significance. The affinities between distributed use of EM methods and collaborative work with spreadsheets [11] suggest that an EM perspective may be well-suited to the addressing the problems that confront HCI within the broad setting of social and organisational structures (cf. Level 3 of analysis in HCI [8,12]).

References

1. A V Aho, J E Hopcroft, J D Ullman *Introduction to Algorithms and Data Structures* Addison-Wesley 1982
2. J Allderidge, W M Beynon, R I Cartwright, Y P Yung *Enabling Technologies for Empirical Modelling in Graphics* CSRR#329, University of Warwick 1997
3. W M Beynon *Empirical Modelling for Educational Technology* Proc Cognitive Technology '97, IEEE 1997, 54-68
4. W M Beynon, Y P Yung, M D Atkinson, S R Bird *Programming Principles for Visualisation in Mathematical Research* Proc 1st Int Conf on Compugraphics, Portugal 1991
5. W M Beynon *Modelling State in Mind and Machine* CSRR#337, University of Warwick 1998
6. W M Beynon, I Bridge, Y P Yung *Agent-oriented Modelling for a Vehicle Cruise Control System* Proc ESDA'92, ASME PD-Vol.47-4, 1992, 159-165
7. M Chmilar, B Wyvill *A Software Architecture for Integrated Modelling and Animation* Proc CGI'89, 257-276
8. K D Eason *Ergonomic Perspectives on Advances in Human-Computer Interaction* Ergonomics 34(6), 721-41
9. I Hacking *Representing and Intervening: Introductory Topics in the Philosophy of Natural Science* CUP 1983
10. William James *Essays in Radical Empiricism* Bison Books 1996
11. B Nardi *A Small Matter of Programming: Perspectives on End User Computing* MIT Press 1993
12. J Preece et al *Human-Computer Interaction* Addison-Wesley 1994

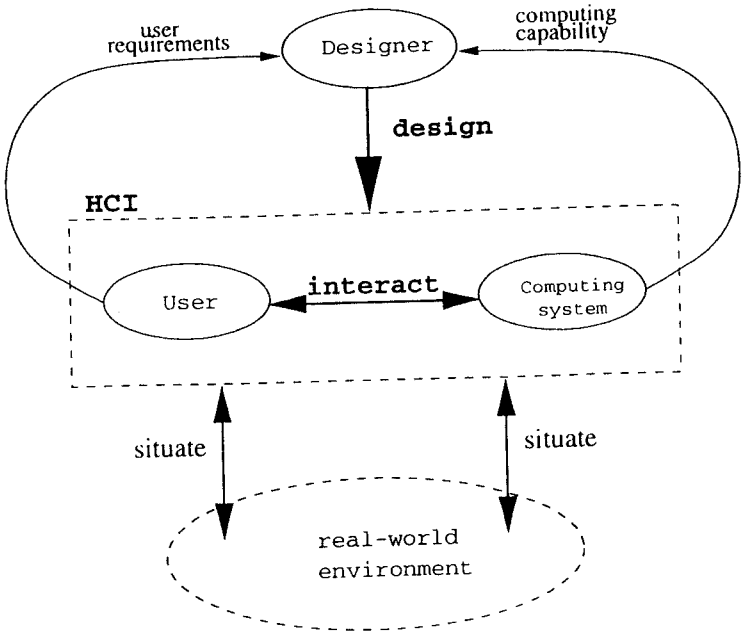

Figure 1. Conventional HCI Design

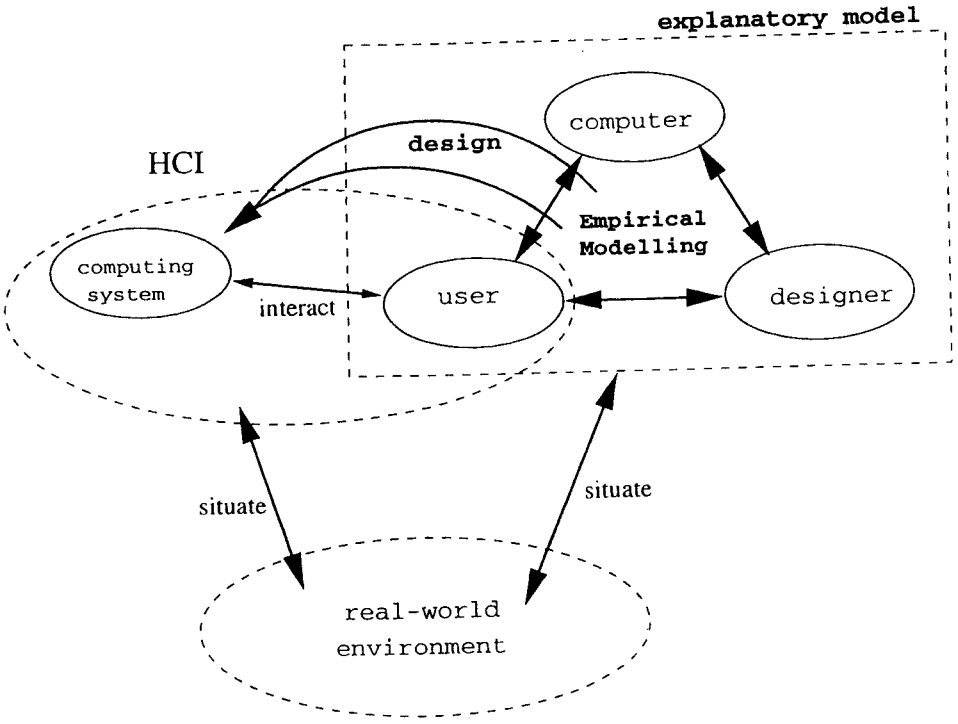

Figure 2. HCI Design in an Empirical Modelling Framework

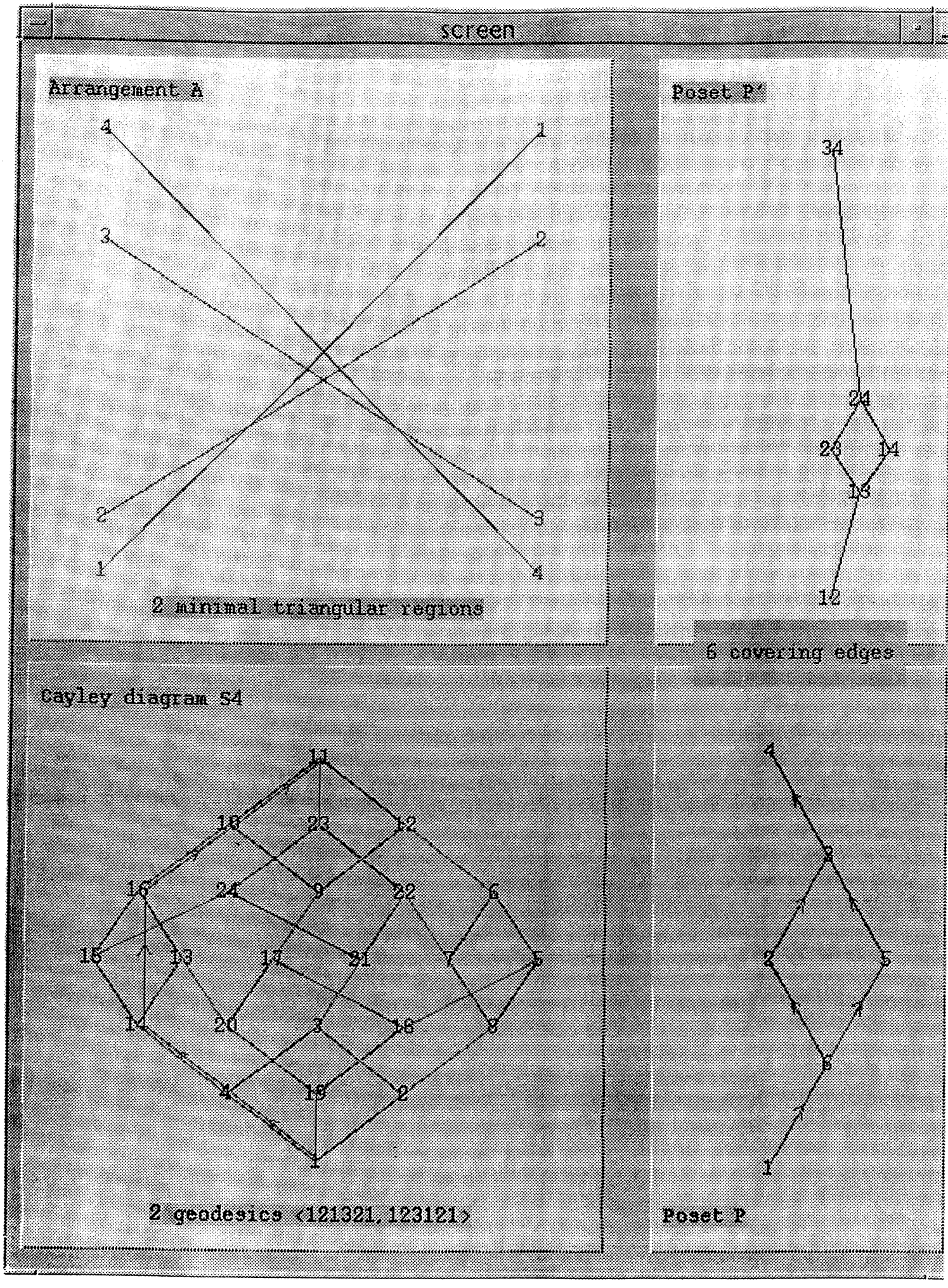

Figure 3: Visualisation for a configuration of 4 lines

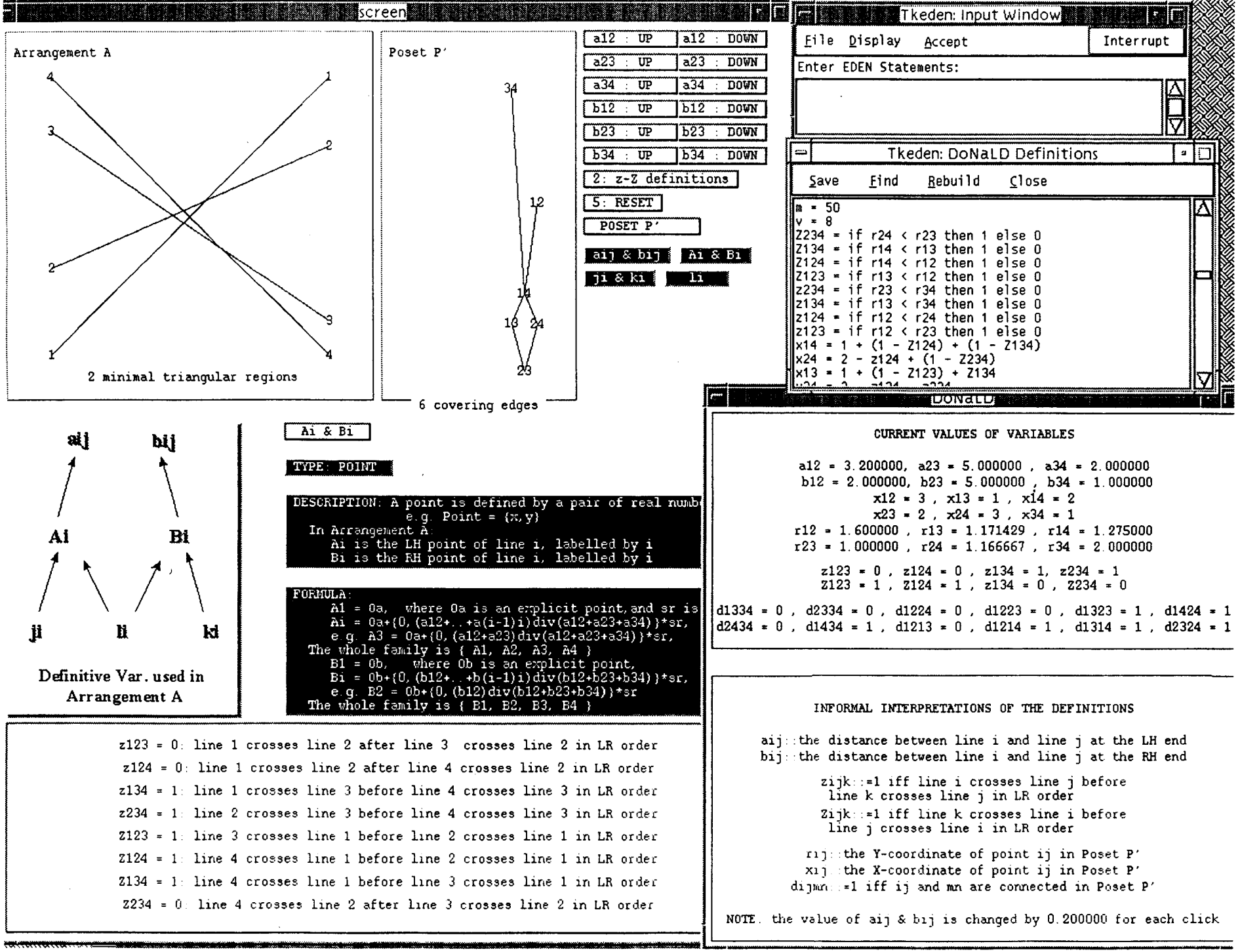

Figure 4: Rungrattanaubol's extensions to the *lines* model

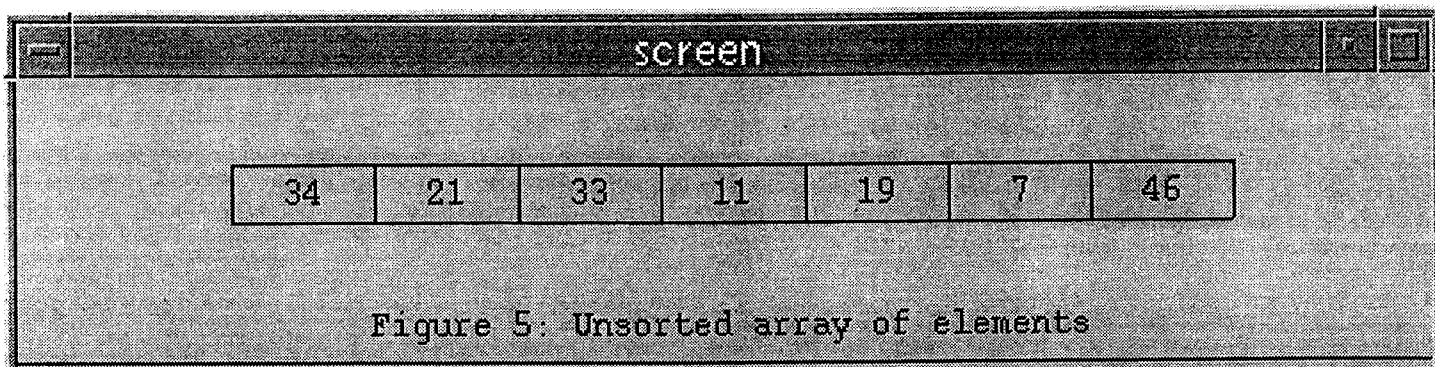

Figure 5: Unsorted array of elements

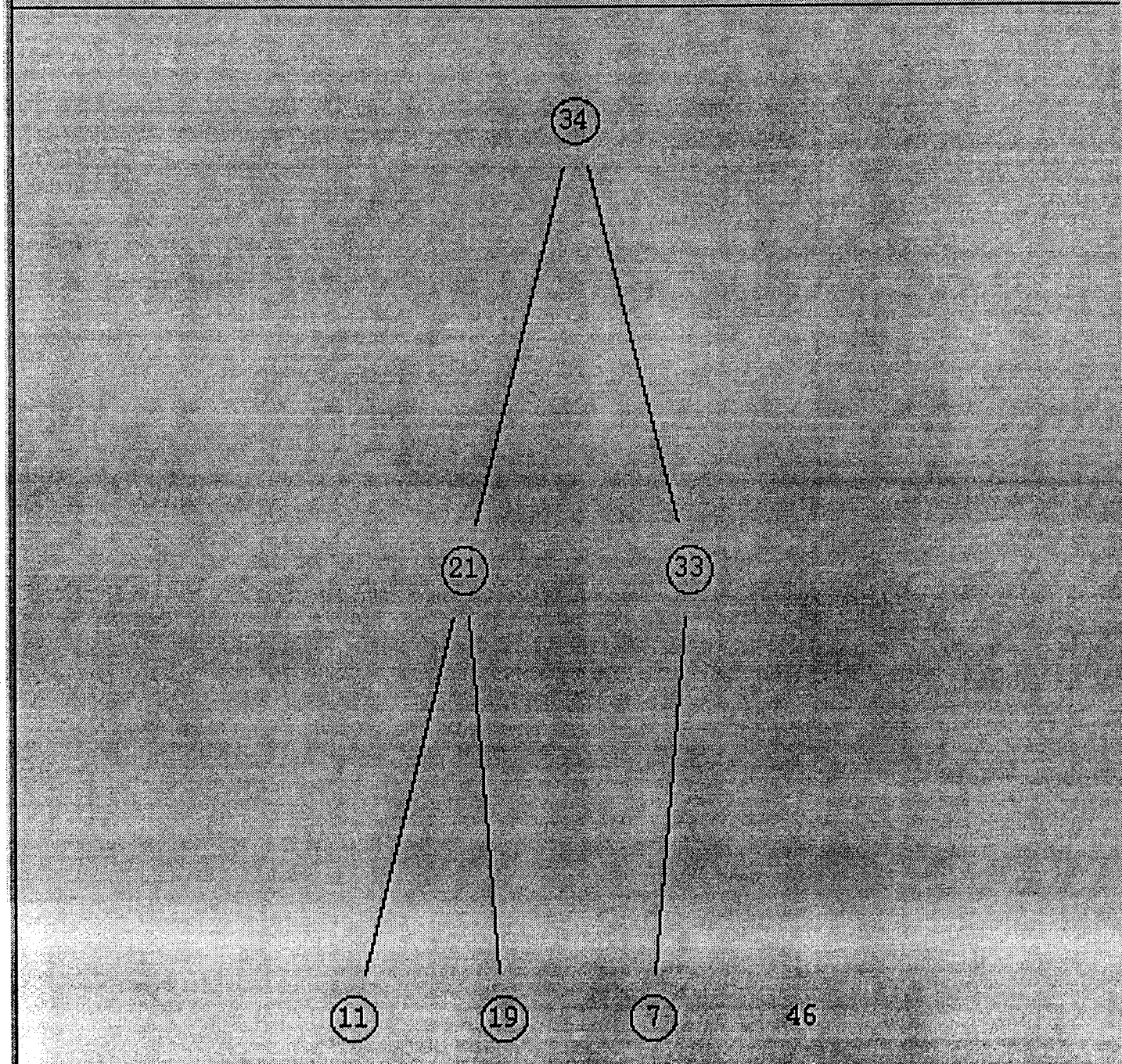

Figure 6: Sorted array extraction - cf. the array in Figure 5