The Skull beneath the Skin:
Entity-Relationship Models of Information Artefacts

T. R. G. Green

> MRC Applied Psychology Unit
> 15 Chaucer Rd, Cambridge CB2 2EF
> thomas.green@mrc-apu.cam.ac.uk
> http: //www.mrc-apu.cam.ac.uk/personal/thomas.green/

D. R. Benyon

> Dept. of Mathematics and Computer Science
> Open University, Milton Keynes MK7 6AA
> d.r.benyon@open.ac.uk

Abstract

Data modelling reveals the internal structure of an information system, abstracting away from details of the physical representation. We show that entity-relationship modelling, a well-tried example of a data-modelling technique, can be applied to both interactive and non-interactive information artefacts in the domain of HCI. By extending the conventional ER notation slightly (to give ERMIA, Entity-Relationship Modelling for Information Artefacts) it can be used to describe differences between different representations of the same information, differences between user's conceptual models of the same device, and the structure and update requirements of distributed information in a worksystem. It also yields symbolic-level estimates of Card et al.'s (1994) index of 'cost-of-knowledge' in an information structure, plus a novel index, the 'cost-of-update'; these symbolic estimates offer a useful complement to the highly detailed analyses of time costs obtainable from GOMS-like models. We conclude that, as a cheap, coarse-grained, and easy-to-learn modelling technique, ERMIA usefully fills a gap in the range of available HCI analysis techniques.

1. Introduction

We present a novel technique for analysing interactive devices and non-interactive notations (together called here 'information artefacts' and discussed in detail below). The technique is built on the observation that the structure of an information display closely resembles the structure of a database: like a database, a display usually contains several types of item with several instances of each type, and there are relationships between the items. Each structure can be realised in many different ways, which may look very different, yet since these 'renderings' preserve identical underlying structures there are many important properties which remain unchanged in the different possible versions. The function of our data-centered analysis is to reveal these invariant properties: the skull beneath the skin (Eliot, 1958).

There are now many analytical techniques available in the HCI world. The best-known is undoubtedly GOMS (Card et al. 1983), and another well-known one is User Action Notation, or UAN (Hartson and Hix , 1989). Each of these focuses on the details of the user's interaction with a device, specifying the precise actions required – down to the level of individual mouse clicks – to achieve a particular result. Analyses at this level are essential to HCI, yet they are not the solution to all problems, because they apply to the 'rendering' of the information, that is, to what the user sees rather than to the underlying structure. Other types of analysis deal with more person-centered issues, such as the perceived connotation of the artefact, or its place in the total worksystem; again, this type of analysis is essential to HCI, but does not present the complete story.

The analysis method we shall describe is derived from entity-relationship modelling, augmented to give ERMIA (Entity-Relationship Modelling for Information Artefacts). ERMIA complements such surface-oriented approaches as GOMS. It, too, presents only part of the story, but it has different advantages and can reveal different insights from surface-oriented methods. Firstly, putting aside the rendering means that our method can be used in different ways from previous methods:

- It allows us to analyse non-interactive structures, such as notations and tables, in the same 'language' as interactive ones. In contrast, a system like User Action Notation is little use when the user makes no actions.
- It can be used at an early stage of design, long before the final rendering has been decided on.
- It can be used to compare possible similarities between information artefacts that would have been masked by differences in the renderings.

Secondly, since we can analyse the structures of mental models in the same terms, our method gives insights into little-explored areas:

- It allows different mental models to be compared (designer's/user's, or across different users).
- It allows the structure implied by the artefact to be compared to the mental model held by the user.
- It allows distributed systems to be analysed, i.e. worksystems in which requisite information is distributed across different people and/or artefacts.

Finally, since it is easy to describe algorithms for traversing the structures in our formalism, this method gives a symbolic metric of effort required to search or change a structure:

- A symbolically-evaluated index of 'cost-of-knowledge' can be computed as part of a design evaluation.
- A cost-of-update index can also be computed; this is a novel concept, to our knowledge.

Aims and structure of the paper

The aim of this paper is to give an informal description of ERMIA and to show how it can be used in the ways just described, reserving a more rigorous description for a subsequent paper (Benyon and Green, in preparation). In the following section we shall first describe the reasons for choosing to use entity-relationship modelling, and then describe ERMIA's slight extensions to the traditional version, needed for our purposes. Section 3 describes how ERMIA can be used to compare mental models with each other or with implied models, and how distributed cognition can be analysed. Section 4 investigates symbolical evaluations of cost-of-knowledge. Section 5 presents our conclusions.

2. The ERMIA formalism

2.1 Information artefacts as the target for HCI

Previous methods of analysis have concentrated heavily on interactive devices, as we have indicated. Given the widespread effects of computer-based technology that is understandable, but unduly narrows the scope of enquiry. It is as though the structure of a calendar is part of HCI when the calendar is made interactive (as in the Spiral Calendar system, by Card et al., 1994, described below), but is part of some other discipline, not part of HCI, if the same calendar is available as a paper-based artefact.

We have adopted the *information artefact* as the target of our research, defining it as any artefact whose purpose is to allow information to be stored, retrieved, and possibly transformed. Interactive devices like spreadsheets, word processors, and music notators are clearly examples of information artefacts, but so are non-interactive devices like tables, documents and musical scores[1]. Thus, when a table is prepared on a spreadsheet, there are two levels of interest: the table itself, and the interactive system.

Tables, documents and other non-interactive information artefacts possess a clearly apparent structure, usually lending itself well to ERMIA's type of modelling. In many cases, that is the most important part of the artefact to model, even in interactive systems; its structure determines the usability of the system as a whole in many important ways that are not made apparent by the surface-oriented methods of analysis. In the case of a design for a calendar, for example, the time taken to find a particular entry (the 'cost-of-knowledge', in Card et al.'s clever phrase) obviously depends on the structure of the calendar, as well as on the form in which that structure is presented. A technique like GOMS can be used to estimate the access cost for a given structure, but what GOMS describes is the procedure for accessing an entry – it does not describe the actual structure.

---

[1]One could perfectly well argue that paper is an interactive medium of sorts, but we are following common usage.

It is less apparent that there is a structure at the level of the interface to spreadsheets, word-processors and other interactive devices. Nevertheless, by imagining a different type of interface to the same data, we can see that two levels exist. The data and formulae of a spreadsheet could be arranged in lines down the page; the text held in a word-processor could be displayed in a hierarchical data structure of paragraphs, sentences, words, and letters, and might in principle be viewed one letter at a time, just as database system can display one record at a time.

2.2 The parallel between data structures and information artefacts

Entity-relationship modelling is a well-known data-modelling technique, widely used as a design tool for record handling. In its traditional form it is intended to promote the design of efficient computerised systems operating on conventional sequential hardware, and there are techniques for optimising structures with respect to computer operations. For our present purposes, however, we need to analyse structures with respect to human abilities rather than computer ones, so we shall present an enriched version that we call Entity-Relationship Modelling for Information Artefacts (ERMIA).

Distinguishing between structure and content is a hoary problem. A simple-minded description of structure is that if you look at just one page of a telephone directory you can probably invent other fragments which have the same structure – a list of subscribers ordered by surname, associated with an address and a number – even without knowing what the content should be. At any rate, when we describe a structure, we wish to do so without encumbering ourselves with details of its content. So the problem is to describe an information artefact without mentioning every instance of similar objects (such as names or numbers) and without including unwanted detail about physical appearance (but able to include it if thought desirable).

Entity-relationship (ER) modelling solves this problem by analysing a structure into constituent *entities* (e.g. Name), *attributes* that distinguish between different entities and between different instances of the same entity, and *relationships* between entities.

Why ER?

There are several systems for data modelling, any of which could have been a possible candidate for our purposes. Entity-relationship modelling was one of the first data modelling techniques to be developed (Chen, 1976) and has become very popular, with numerous texts introducing it for the use of systems analysts and computer scientists (Benyon, 1990; Howe, 1981). Unlike data structure diagrams (Bachman, 1969) or the relational model (Codd, 1982) which focus primarily on implementational structures, ER diagrams are conceptual models, distant from any given implementation. It is central to most system development methodologies such as SSADM (Downs et al., 1988) including some modern object-oriented methods (e.g. Jacobson et al., 1993). Thus, by using this notation, we are buying into a method that is already common currency among commercial designers, instead of hoping that they would find time to learn a new formalism.

One of the main reasons for using ER modelling is that it enforces extreme simplicity. In comparison to the much more powerful first-order logic, ER modelling is very weak – but it is very much easier to learn than the predicate calculus.

Another reason is that ER is conventionally presented in graphical form, which is thought to improve comprehensibility. Although the value of diagrams should not be taken for granted, they have been shown to be beneficial in other forms of problem solving and design (e.g. Carroll et al., 1980; Schwartz, 1971), and Batra and Antony (1994) showed that trainee designers using ER notation performed more slowly but more correctly than ones using a text-based relational model. We shall adopt the graphical convention, but of course an equivalent textual form is easily provided.

2.3 Entities and relationships in interactive information artefacts

ERMIA uses an adaptation of entity-relationship modelling (about which we shall have much more to say) to describe structures. In interactive systems, such as simple drawing programs (Figure 1) there are two kinds of structure: first, the drawing itself, or whatever is being developed, and second, the interface.

Structure of drawing: In the example we have chosen that structure is very simple, because the drawing just consists of graphic objects, and any object can be changed without affecting any other one – hence there are no relationships between objects.
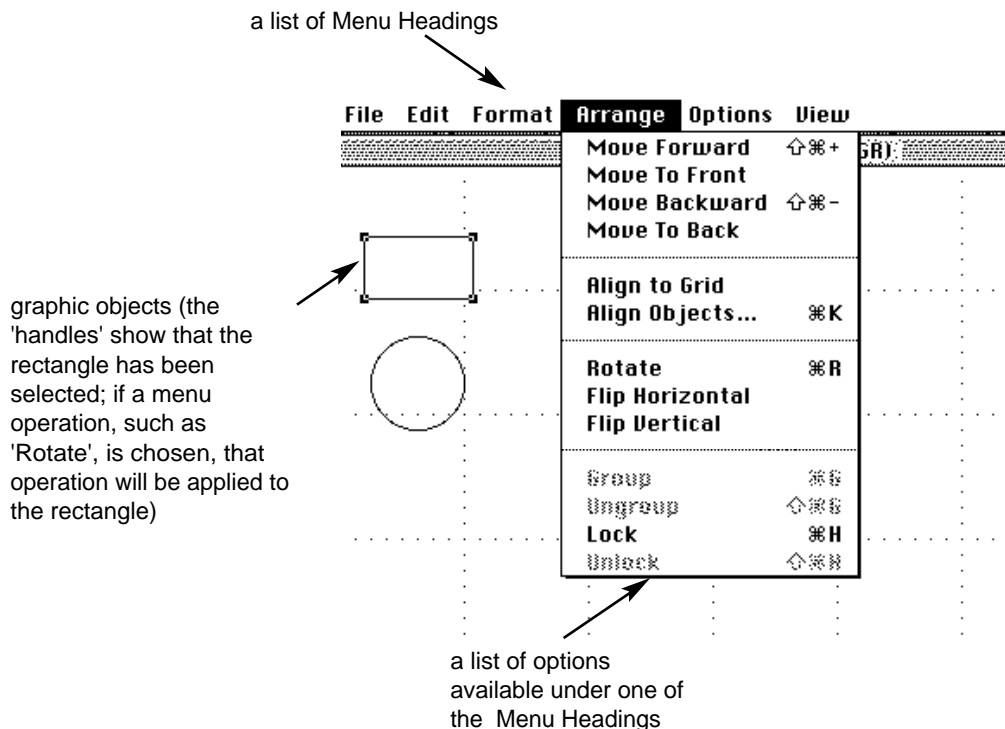

Figure 1: A simple drawing program, showing the document being created (a drawing, currently consisting of a rectangle and a circle) and the interface to the application

In ERMIA, we would say that drawings have a structure comprising one kind of entity, called a Graphic Object, which has two attributes. One attribute is its shape, the other is whether it has been selected or not. In Figure 1 the rectangle has been selected but the circle has not been. Other attributes that could have been modelled include position and size; the modeller has chosen to ignore these.

As a foretaste of ERMIA's graphical notation, Figure 2 shows the ERMIA map describing such drawings. Notice that although the drawing itself contains two graphic objects, a rectangle and a circle, the map only contains one kind of entity, the generalised Graphic Object. The map is a list of all the different kinds of possible things and how they are related; it is not a list of the particular things that turn up on an individual drawing.
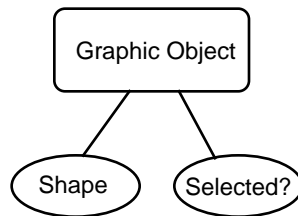

Figure 2: The structure of simple drawings

Interface structure: This structure is more complex. Menu systems have two kinds of entity. One kind of entity comprises the various menu headings, such as File, Edit, and Arrange, and the other kind comprises the various items that are found under the headings, such as 'Save'. Both kinds of entity have a name, which counts as an attribute. More interestingly, there is a relationship between the two kinds of entity. Can we imagine an interface that contains a menu item without a menu heading? No, because there would be no way to get at it. So we conclude that every Item must be associated with a Heading. On the other hand, we can imagine a menu which contained no items: not much use, but not impossible, especially while the software is being developed.

It is also important to note that each menu heading can list many items, while each item is normally only found under one heading – in other words, the relationship of Heading to Item is 1:many (written 1:m or 1:M). Is it strictly true that the relationship is 1:M? Not quite. It was by being forced to consider the question precisely that the authors became aware that different pieces of software are based on differing interpretations of the Apple Macintosh Interface style. There is actually nothing to prevent the same menu item being listed under more than one heading. So an item like 'Format' might be found under a Text heading and also under the File heading; the true relationship between Heading and Item is therefore many to many, or M:M as it is written. But some applications (notably, applications for driving other applications) get into difficulties if two items have the same name, because only one of the two can ever be accessed. These applications have been designed with the assumption that the relationship is 1:M.

The difference between the two possible interface structures is small but sometimes profound. Probably at the time of designing the style guidelines for the Macintosh interface it was overlooked. Drawing ERMIA maps forces the designer to be explicit; the map for the 1:M version is shown in Figure 3.
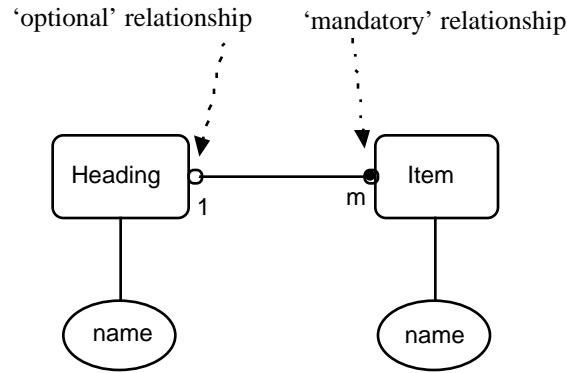
Figure 3: ERMIA structure of a menu system containing Headers and Items. The relationship between Heading and Item is 1:M (that is, each Heading can refer to many Items, but an Item can only be associated with one Heading). For Items, the relationship is mandatory (that is, every Item must have a Heading), but a Heading can exist with no associated Items.
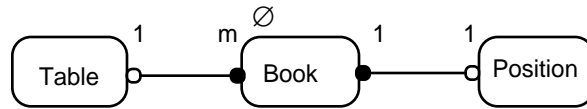
2.4 Internal structure of entities

To adapt ER notation for our purposes a little extra machinery will need to be added. Conventional entity-relationship modelling was deliberately intended to abstract away from details of the implementation, to allow the data-modeller to concentrate on issues of defining records and data in an unambiguous way. Issues such as complexity of search were not part of the consideration at that level of analysis, so the manner in which data was stored could safely be left out of the model.

In the consideration of information artefacts, however, the issue of search time and complexity has great importance. Timetables, telephone directories, and dictionaries are hand-held repositories of data that would be unusable unless they had been designed for effective search. Computer-based repositories also need to enable effective search.

In order to permit reasoning about searchability and search costs, we shall re-introduce some of the implementation details normally omitted from conventional ER models, although we represent them in a very simplified and abstract way. We do this by distinguishing between different types of entity class. An entity class (perhaps better thought of as an entity *store*) stands for many occurrences of the entity and the whole purpose of an information artefact is to be able to locate one particular member of the class. The internal structure of the entity class may give the user more or less assistance in that. We distinguish five types. Each has its own characteristics of time and memory requirements when it forms part of a search for information.
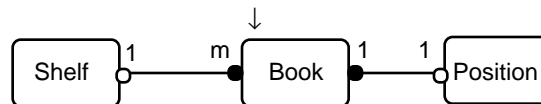
i: Pile. To illustrate the different types of entity store in concrete form we shall consider what can be done with a very familiar information artefact, namely, a collection of books. The principal entity here is obviously Book, an entity store standing for all the individual books. We can start with the case where the books are haphazardly strewn across a table, so that the entity Book contains no internal structure whatsoever. We call this kind of entity-store a *pile* and mark it in the diagram with a phi, ∅:

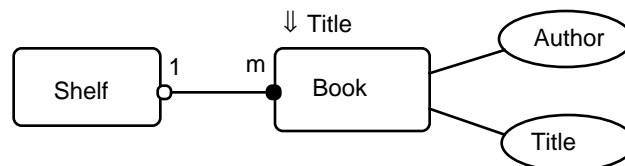## Figure 4: Books heaped on the table form a pile structure.

If there are N books, then on average it will take N/2 inspections of books to find a specified book. And as the books are piled haphazardly on the table, we shall need to remember which books have been examined already (or, equivalently, which positions have been examined already); that entails remembering N/2 items. In the worst case, someone else is also examining the books and putting them down in different places, so that we cannot even rely on remembering the places we have looked; we **have** to remember each individual book.

ii. Chain: By putting the books on a shelf – even if unsorted – more structure is introduced, which gives the seeker far more help. He or she will still need N/2 inspections on average, but the number of positions to be remembered has been reduced to 1, the current position. We call this structure a *chain*, and represent it with a single arrow.

Figure 5:   Books at random on the shelf form a chain structure.

iii. Sorted list: If the books are arranged in alphabetical order by title, search can be still faster. The seeker can make a guess whether to start in the middle of the shelf or at one end, depending on whether the title sought starts with an early letter or a late letter.   He or she might even adopt a 'binary chop' algorithm – start in the middle, then move halfway to one end, then move a quarter of the distance, and so on. This type of entity store is called a *sorted list*, represented as follows. (Notice that it is the occurrences of the Book entity which are ordered by Title as indicated. If shelves are ordered by subject, say, a similar symbol and annotation is appended to the Shelf entity.)

Figure 6: Books on the shelf in alphabetic order by Title

iv. Unsearchable: Now let us suppose that an unsympathetic library system has control of all the books, and that they are all locked away in stacks to which we are denied access. To get a book, we have to find the details and give them to the librarian, who will fetch the book. From the librarian's point of view, the books may be arranged in any of the structures above; from our point of view, however, they are

unsearchable. They can only be accessed via some further repository such as a catalogue. An unsearchable entity store is marked with a cross, X:
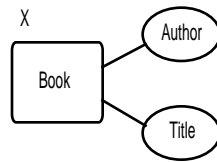

Figure 7: Books locked in the stacks, from the user's point of view. There is no Shelf entity because the user has no way of knowing whether the books are on shelves or lying around in heaps on tables, etc.

v. Hashed: Lastly, suppose that the books are few enough and familiar enough for the user to be able to recall instantly the position of each book. This corresponds to the notion of a hash-coding in computer science, where an address is computed directly from the content, allowing the item to be retrieved with zero search. For this structure, which we shall not encounter frequently, we use the obvious symbol #.

To summarise: we distinguish five main types of entity-store: pile, chain, sorted list, unsearchable, and hashed. The internal structures dictate the expected number of search steps to find a target, and the number of items that have to be remembered. Many of the differences between the usabilities of different information artefacts are consequences of these differences. Search can be made easier, however, if there are perceptual cues to help.

2.5 Perceptual attributes and conceptual entities

Our examples so far have ignored important differences. Attributes that differ through perceptual aspects, such as shape or colour, can be recognised or discriminated faster than attributes differing in their symbolic labels, which have to be read (e.g. the titles of books). Nor have we so far indicated which entities belong to the external world of physical objects and which belong to the internal world of conceptual entities.

ERMIA makes a distinction between entities with perceptually-coded attributes and ones without, by a thick line around the attribute oval (Figure 8). A search process can be much faster if a perceptually-coded attribute can be used ("It was a red book").
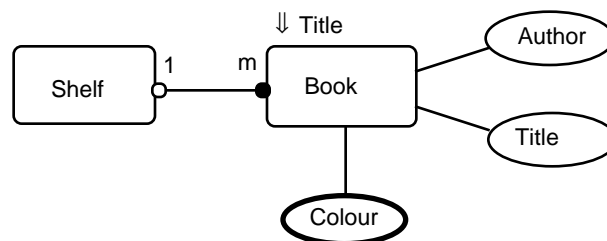

Figure 8: Because colour can be perceived directly, it is shown as a perceptually-coded attribute.

Lastly, ERMIA introduces a distinction between entities that are directly represented by some part of the information display and entities that are not, for example the plot of a book (boy meets girl / Cinderella makes good / hero defeats evil / hero is destroyed by own personality, etc.). These conceptual entities are

represented by dotted lines. Conceptual entities are always hash structures with zero search time (Figure 9).
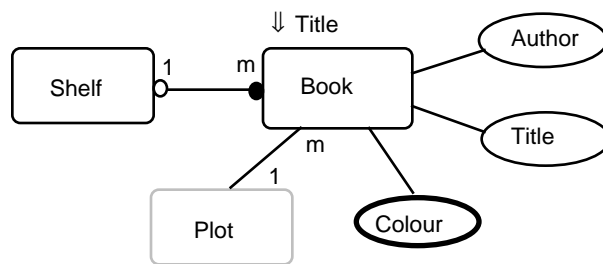

Figure 9: Plot is a conceptual entity, marked by dotted lines.

There is nothing to prevent conceptual or perceptually-coded entities from taking part in the usual relationships: for instance, plot types could be colour coded, making it easier to find a book with a given plot type (Figure 10). This takes us to the limits of conventional ER's expressive powers; there is no way to indicate within the formalism that Book and Plot have the same colour, or that there is a 1:1 mapping between Plot and Colour; if it is necessary to resolve such semantic ambiguity, the diagram would have to be annotated or the notation would have to be enriched (see Verheijen and van Bekkum (1982) for one method).
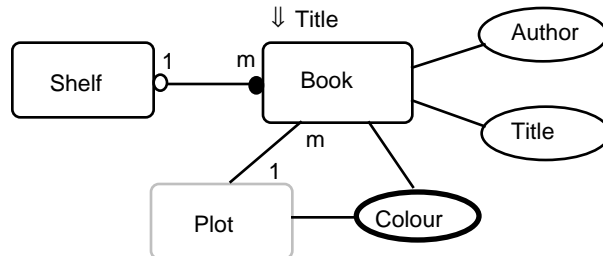

Figure 10: The plots are colour coded.

When the relationship between a manifest entity, such as Book, and a conceptual entity, such as Plot, is mapped onto a perceptually-coded attribute, such as Colour, the effect is that a form of hash coding is introduced – books with a given plot type can be accessed with near-zero search.

3. Structures, representations, and models

When the 'same' information is made available in two different representations, what is it that remains the same? One answer is that the *conceptual* information remains the same. ERMIA models provide a link between the conceptual information that is stored in an information artefact and the physical way that the information appears to someone using the artefact; or to put it another way, a link between the information to be displayed, and the display itself. That display may not succeed in conveying all the entities, attributes and relationships that are needed by a user; or the information may all be presented, but the presentation may be structured in way to make the information hard to use.

Note that the conceptual structure has no privileged status of objective truth. When we speak of 'the' conceptual structure, we mean that structure that we, the authors, have intuitively worked out. Other people may have different conceptual structures (and indeed, better methods of knowledge elicitation might have revealed that we ourselves have slightly different conceptual structures to the ones we describe). The techniques of accurate elicitation will not be discussed here.

It is well-known that different users may form different ideas or 'mental models' of how something works, or that the designer of a system may not succeed in imparting the intended view to the users; but although it is well-known, the tools of HCI have not included a compact representation of the differences. In this section we show how different ideas can be represented in ERMIA.

3.1 Same information, different representations

It will be convenient to think of any particular information artefact as a *viewport* onto a conceptual information structure. Different ways to present the same conceptual information can then be regarded as different viewports onto the same structure. Each viewport has its own structure, and by definition that is a *manifest* structure – the viewport is defined by the relationships and entities that it displays. Conceptual information that is omitted from the viewport can be represented as conceptual (dotted-line) entities, or in extreme cases a separate diagram may be necessary to show how the viewport's structure relates to the conceptual structure.

In the following example we compare the structure of two viewports onto the same information: a visual version of a train timetable and a 'talking' timetable.

Consider the working of passenger trains over a short railway with three stations, Camelot, Atlantis, and Lyonesse. In the authors' conceptual structure, the workings comprise a sequence of train movements, where in each train movement a locomotive and some carriages travel from station to station. Movements are uni-directional, so after a train has travelled from Camelot to Lyonesse its return trip is a new movement. Movements presumably have to have an identifying attribute which we shall call Train ID, just as airline flights have a Flight Number.

Figure 11 shows part of a typical timetable. Each column describes a single train-movement. The Train ID is not normally shown in public timetables, at least in Britain, so it is a conceptual entity, not a manifest entity, and must be shown as such on the ERMIA model.

Camelot	9.05	14.18	17.37
Atlantis	12.13	------	20.41
Lyonesse	14.36	18.22	22.38

Figure 11: an example train timetable.

The ERMIA model describes the inter-relationship between the rows and columns via the Cell entity (each Row has many Cells, each Column likewise); and it also shows that rows correspond to stations, and cells to times. The columns correspond to the Train ID, shown as a conceptual entity because it has no direct representation in the table.
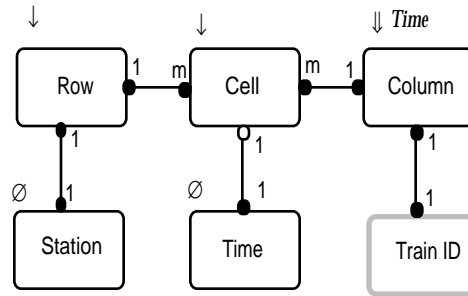
Figure 12: structure of a paper-based timetable.

How to search a paper timetable:
Suppose we wish to discover a train that will take us from Camelot to Lyonesse to arrive before 21.00, without departing earlier than necessary. The search program will be something like this:

> find the appropriate row by searching down left-hand column for a cell containing 'Lyonesse'
>
> search cells within that row and find first cell-within-row greater than 21.00
>
> move back to previous cell
>
> move to top of column; time in that cell is the target (in this case, 14.18)

To find the desired train departure time we need only remember the target station and time, plus a single item for location within the table.

The talking timetable has a very different structure. There are many different structures that could be adopted. We have based our example on an actual recorded timetable:

> "This is the talking timetable for trains between Camelot and Lyonesse, calling at intervening station Atlantis. The first train of the day leaves Camelot 9.05 am arriving Lyonesse 14.36; subsequent departures are at 14.18, arriving 18.22, non-stop, and 17.37 arriving 22.38."

The main component of the talking timetable is a 'record' with this structure:

> Train-movement
>> Departure – origin plus time
>> Arrival – destination plus time
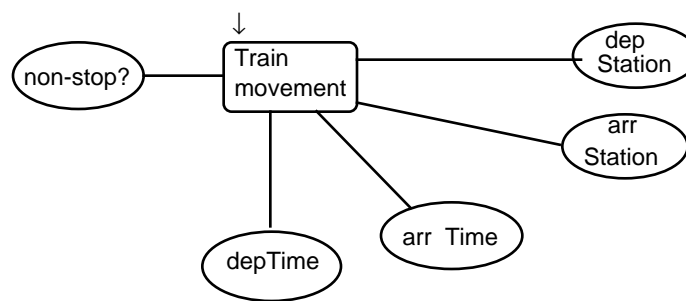>> Whether non-stop

This gives the following structure.

Figure 13: Structure of a talking timetable.

How to search a talking timetable:

The structure of the single entity-store, Train-movement, is a chain, so we have to go through the instances one by one. To find the desired train we have to wait until we hear the first arrival time after 21.00, then we have to recall the information about the previous train, so we have to store all those items in memory. Thus, the load on working memory is considerably greater than for a paper timetable.
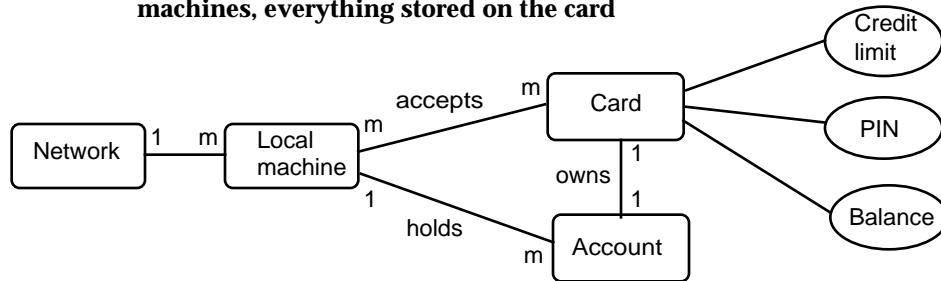
3.2 Describing different users' models

In the previous section we showed how the conceptual information could be distinguished from a manifest representation supplied by a particular viewport. Constructing ERMIA models that *only* represent conceptual information gives us a method for comparing users' mental models. There are numerous other formalisms for knowledge representation, of course. The advantages claimed for ERMIA are its simplicity, and more importantly its ability to represent different device structures in the same language.

One of the few comparative investigations of users' models in the wild was made by Payne (1991), who elicited models of the functioning of automatic teller machines (ATMs) from 16 subjects and showed how they affected people's behaviour in using the machines. He reported:

> "A striking observation about the mental models of S14 and S15 .. is how different they are. One subject [S14] believes that a great deal of information is stored on the plastic card, and that this information is updated when the machine is used. The other subject [S15] believes that the only information on the card is the user's PIN (Personal Identification Number). The first user believes that each bank machine is 'intelligent' ... The second believes that each bank machine is a 'dumb' terminal to a central computer, where all the records and stored and all the computations are performed. ... Such variety is rampant in the data." (p. 12)

Like any natural-language description of this type of material, Payne's description has to find a compromise between precision and tedious verbosity. Figure 14 shows how the structural aspects of these different beliefs can be represented in ERMIA. Because ERMIA presents a clear view of the different models, it could be used as part of the process of reasoning about the revelation of models. If we have a designer's model that the designer wishes to reveal, he or she can look at the model of the interface and see to what extent the 'intended' model shows up. Similarly one can gather different user views, in the manner of Payne's work, and compare them to the designer's view, making the models and their possible differences explicit through ERMIA.

Subject 14: network of interconnected intelligent machines, everything stored on the card

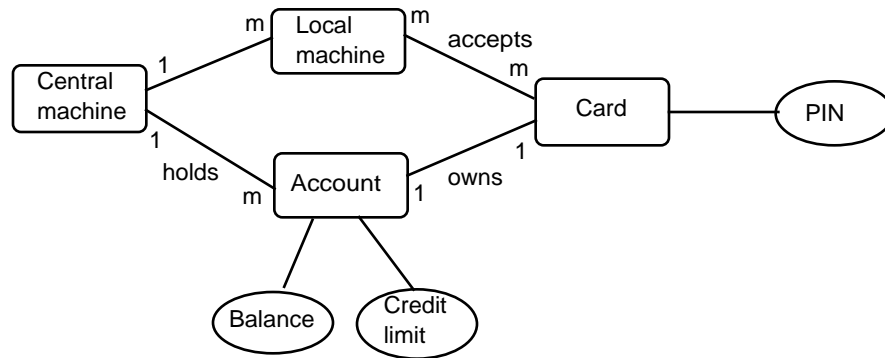**Subject 15: central machine with local 'dumb' clients, nothing on the card except the PIN**

Figure 14: Comparison of two mental models of ATMs described by Payne (1991)

3.3 Distributed information

In many worksystems information is not held in any one place, nor by any one person; but rather it is distributed through the worksystem (see, for example, Hutchins, 1994). Although this is frequently referred to as 'distributed cognition', in the present context where information may be held by artefacts rather than by people we shall use the less exacting term 'distributed information'.

ERMIA models of such systems locate the sources of information and thereby identify some parts of the work practices that will be needed for the system to function successfully, including the problems of maintaining and repairing the system when new information arrives or when a component is lost. We have not seen other techniques that make that explicit.

Smart telephones are a simple example of distributed information. These devices contain a list of short-cut codes, so that when some code sequence like #1 is keyed, the phone actually dials a full-length number. As a rule, they do *not* contain the information about who that number calls. It is the user's respon-sibility to record the relationship between names and dialling code sequences. Since surprisingly many designs have no provision for externalising that information, we have shown it in the model as located in the user's head. In practice, such designs are therefore frequently disfigured by sticking dialling code

lists onto them, in true 'back of envelope' style. Some telephone handsets do actually contain a small notebook to record the relationship. Either way, the ERMIA model makes explicit the problem of how to store and access this information.

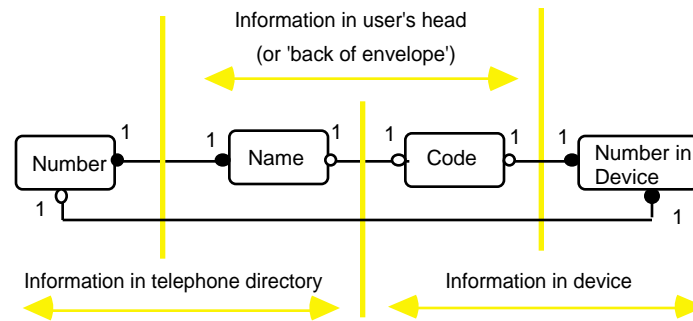Considered as a worksystem, the overall structure is as shown in Figure 15.

Figure 15: A 'smart' telephone. Every Name has a Number and vice versa, but only some Names have Codes (and vice versa). The numbers held in the smart phone (Number in Device) all correspond to real numbers, but of course not every real number is stored in the device, hence the relationship 'Number in Device' : Number is optional on the Number's side. Nor does every Code have to be assigned to as Number in the Device.

A location-of-information model of this sort can help predict the consequences of the inevitable system upgrades, which enhance performance at the cost of a period of mild chaos as users discover that resources have been deleted or no longer work. In the typical scenario, a new telephone is supplied, which has not yet been customised and does not contain the dialling codes. Notice that to rebuild the Number:Code relationship in the smart phone, the user will need access to the telephone directory – so let us hope that it did not get thrown away, under the mistaken impression that it was now surplus to requirements.

In the model shown in Figure 16, some of the entities are stored in two different ways. For example, Name is available either in the telephone directory, where it is a sorted list, or as a component of long-term memory, where we assume it is hash-coded and retrievable without cost (see Section 2.4). Such overlapping storage structures are probably typical of many worksystems with distributed information.

The smart telephone is obviously a very simple example of distributed system, but it contains many vital elements:

- Information is held in different locations (some human, some not) and stored in different ways
- Some information is held in more than location, posing problems of consistency maintenance
- Some eventualities, such as replacing one of the artefacts or people, may make unexpected calls on rarely-accessed parts of the system (in this case the telephone directory); work practices must be specified to take account of these eventualities (e.g. the directory must be kept in an accessible place; staff must be told how to update the system).
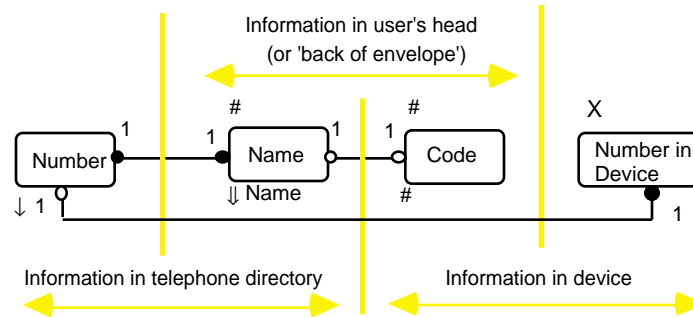
Figure 16: When a new telephone replaces the old one, the Code:Number in Device relationship is lost.

4. Task complexity and the cost of access

It has been claimed that the most important criterion for evaluating information artefacts is the cost of accessing the information: "In a world of abundant information, but scarce time, the fundamental information access task is not finding information, but the optimal use of a person's scarce time in gaining information" (Card et al., 1994). These authors therefore investigated the cost of knowledge in cases where all that the user was doing was seeking information in what they call a 'direct walk of an information structure':

> "We define a *direct walk* to be a task in which a user navigates from a starting point to a goal point in an information structure by a series of mouse points or other direct-manipulation methods. Examples would be the series of mouse clicks and button choices required to operate the Macintosh hierarchical file system or a typical HyperCard stack or many help systems. The essence of a direct-walk is that an information structure is displayed and the user points to, flies to, or gestures over some part of this visible structure resulting in a new display at which time the cycle is repeated until the goal is found." (Card et al., 1994, p. 239)

ERMIA can provide a fast first-approximation measure, to help reason about design decisions. Because ERMIA abstracts away from the physical interface it cannot offer detailed time predictions, but what it can supply is an estimate of the number of user operations (mouse clicks, button choices, etc.) required for locating target information. This estimate will usually take the form "order of N", where N is the number of some kind of entity; or more generally it will be an expression combining several such terms.

4.1 The 'cost of knowledge' for simple searches

Given an information structure it is possible to construct simple algorithms for searching. Although these algorithms are unlikely to be precisely accurate models of human search processes, they yield first-order approximations of the number of search steps required and the number of memory items needed to keep track of the search and to hold the results. A few simple principles determine the costs of knowledge.

Cost of knowledge depends on entity structure

As noted above, the number of steps required to locate a target in a sorted list is proportional to log N, while for a target in a chain structure the number of steps is proportional to N/2, which increases much faster as N increases. The telephone directory, modelled simplistically in figure 17, is a classic example

of everyday search in which the names are a sorted list but the numbers are a chain. In both cases, 1 memory item is needed to keep track of the search.

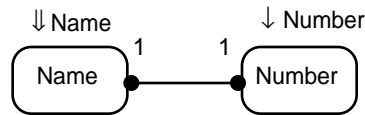⇓ Name   ↓ Number

Name 1 —— 1 Number

Figure 17: a simplistic model of a directory associating names and numbers.

Usually the smaller number of steps is the clear winner, especially for a large information system. Yet there are exceptions. If the search domain is books on a long bookshelf, walking down the length of the shelf takes about the same time whether the titles of books are scanned or not, so there is little to choose between the search methods.

Cost of knowledge depends on target set size
Now consider making a list of **all** the books on all the shelves in a library, where Shelf:Book is 1:M. The task requires searching through instances of Shelf, and then for each shelf, searching through the instances of Book that are associated with that shelf. The number of search steps required to list all the books will be the product

number of shelves * average number of books per shelf

Nested searches are multiplicative
Search processes can be nested indefinitely. If we extend the model of shelves and books to include pages and words (Figure 18), we can search for the title of the first book found for which page 41 contains some target word such as 'floccinaucinihilipilification'. The crucial fact is that we do not know the appropriate shelf in advance, so each shelf has to be searched and then each book within the shelf; whereas we do know the page number, so we can take advantage of the fact that pages are a sorted list.

↓   ⇓ Author   ⇓ Page-No   ↓

Shelf 1 —o m Book 1 —— m Page 1 —— m Word

Title Author   Page-No

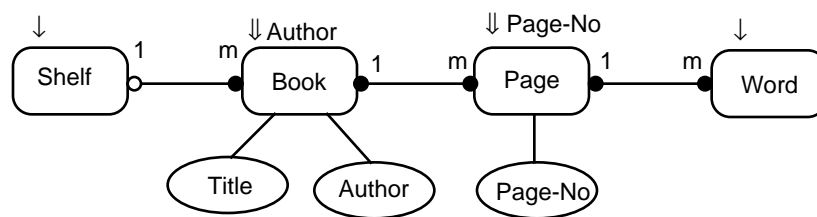Figure 18: Extended model of books on shelves.

The search program will now be something like this:

```
for each shelf S
    for each book B on shelf S
        do binary search for page P in book B using Page-No
            for each word W on page P
                if word W = 'floccinaucinihilipilification'
                then
                    note P and Title (Book B), and stop searching.
```

The number of search steps will be

$$1/2 \; (\#Shelf * \#Book/Shelf * \log \; (\#Page/Book) * 1/2 \; \#Word/Page)$$

The number of memory items to keep track of the search will be 4, one each for Shelf, Book, Page and Word.

Exactly the same analysis could be made for a file system, where Shelf corresponds to Folder (or Directory) and Book corresponds to File. And it is precisely because the number of search steps is so large that power tools for finding files are necessary.

Stages of search are additive

In another familiar type of search, the task is first to locate a subset of entities and then to search inside that subset. These stages are independent searches and their costs of knowledge can be summed. Continuing to work with the example of books on shelves, suppose we want to find where a given word in a given book. So this search has two stages: find the book, e.g. *Robinson Crusoe*, then find the word, e.g. 'floccinaucinihilipilification'.

The search program will now be something like this:

```
1)   for each shelf S
              for each book B on shelf S
                        if Title(book B) is 'Robinson Crusoe'
                        then stop searching shelves.
2)   for each page P in book B
              for each word W on page P
                        if word W = 'floccinaucinihilipilification'
                        then
                                  note P and stop searching.
```

The number of search steps will be:

first search	1/2 (#Shelf * #Book/Shelf)
second search	1/2 (#Page * #Word/Page)
total	1/2 (#Shelf * #Book/Shelf) + 1/2 (#Page * #Word/Page)

The number of memory items will be 2 in each part of the search.

Choice between search methods depends on the implementation

Real life often allows a choice of search methods. To find a name in the telephone book, we could choose to turn the pages one by one, downgrading the entity structure to a chain instead of a sorted list. Or instead of using a simple binary search (open the directory halfway through for a first guess) we could make use of our knowledge about frequencies of names beginning with different letters.

Sometimes real life enforces a particular method of search. A list may be well-ordered but if it is presented item-by-item we have to use serial search, willy-nilly.

When users do not know the structure of the environment, or when they are unaware that power tools can offer different search possibilities, they are liable to make sub-optimal searches, and ERMIA modellers need to be aware of that risk when making estimates of the cost of knowledge.

4.2 The cost of knowledge characteristic function

The *characteristic function* for cost-of-knowledge (Card et al., 1994) describes how much information becomes accessible as a function of user effort.

The concept of the characteristic function will be made clearer by considering a simple calendar system in which a single week is displayed at a time. (We shall call this the Flat Calendar.) If this diary is implemented as a system with a forwards and backwards button (i.e. no other, faster way to get from date to date), then we can see that from a given date, we can reach dates N weeks away in N steps, making the characteristic function is a simple linear relationship between steps and days-accessible. The ERMIA model for the Flat Calendar is very simple:
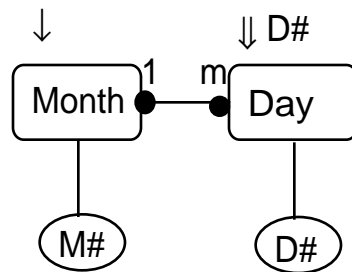

Figure 19: Model for the Flat Calendar. Note that although Month has been modelled as simple chain-structured entity, because the means of access provided by the viewport only allow it to be treated as such. (Conceptually, of course, Month is a sorted-list.)

Card et al. also analysed a device called the Spiral Calendar, in which a given date was selected by successively mousing Century, Decade, Year, Month, Week, and Day. Each mouse click gave an enlarged and detailed view of the particular term, so that clicking on a selected century gave a view of the ten decades inside it, clicking on the selected decade gave a view of the ten years, and so on.
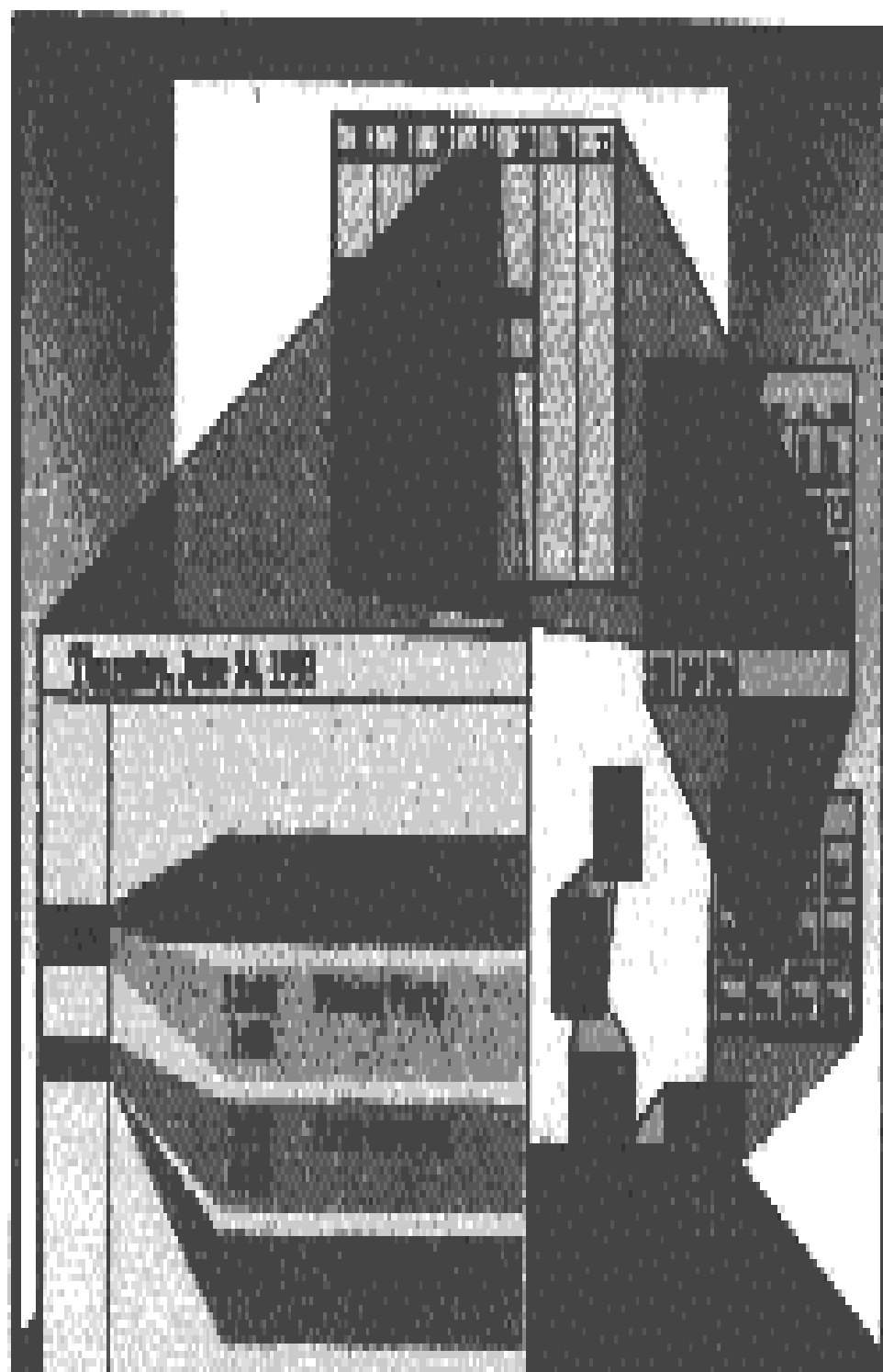
The ERMIA model of the Spiral Calendar is shown in Figure 21.

Figure 21: Model for the Spiral Calendar.

Searching for a given day in the Spiral Calendar is a staged search, because we know in advance the Century, Decade, etc. Each entity is a sorted list, and therefore the number of steps at each stage will be proportional to log N. The shape of the cost-of-knowledge characteristic function is therefore almost exactly logarithmic. Figure 22 shows the number of days accessible for a given number of search steps in the two calendars; Figure 23 plots the curves.

steps	flat	spiral	
1	7	7	week
2	14	30	month
3	21	365	year
4	28	3650	decade
5	35	36500	century
6	42	365000	millennium
7	49	3650000	era

Figure 22: The number of days accessible in a given number of steps, starting from a given day, in the two types of calendar.

Figure 23: The Spiral Calendar's cost-of-knowledge is nearly linear when plotted on log-linear coordinates.

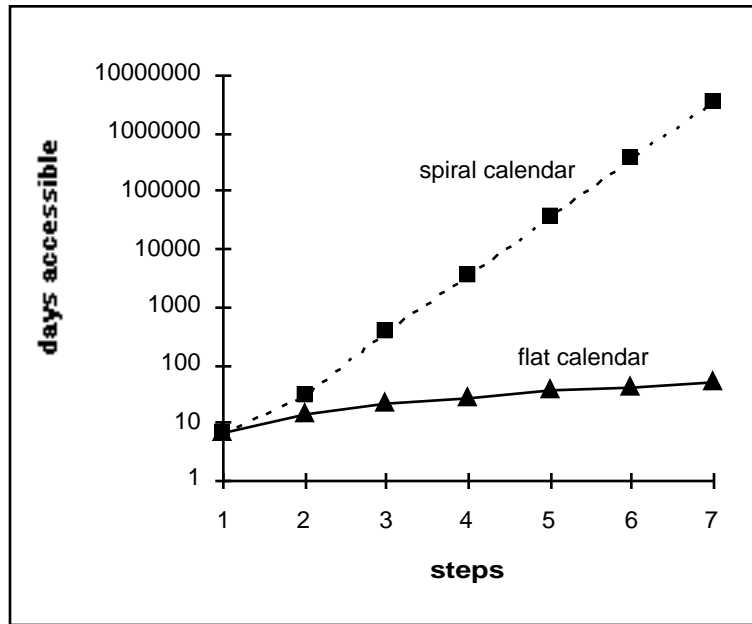These analyses show that the spiral calendar has a lower cost-of-knowledge when the requirement is to access a day that is far distant from the current day. Card et al. used their analysis to demonstrate that fact; our claim is that the same conclusion can be reached much more swiftly by the simple means of making an ERMIA model. They also used their analysis to find ways in which to improve the design of the spiral calendar by reducing the time taken for particular search steps, and for that purpose, ERMIA would not go far enough; the detailed analysis of times that they made would be needed. Thus this example shows the strengths and limitations of ERMIA vis-à-vis GOMS.

4.3 The 'cost of update'

When an information structure is changed, it is sometimes necessary to perform subsidiary actions to restore consistency, as well as the main action. Green (1990) referred to this as 'knock-on' viscosity, but 'cost of update' is consistent with the foregoing topic, 'cost of knowledge'. An example familiar to academic authors is the cost of updating a list of references to a paper-in-progress when a new citation is made. Two of the principal conventions for citing references are shown in Figure 24, which we shall call the 'nominal' and the 'numerical' styles.

style	citation	bibliographic entry
nominal	... see Miller (1956) ...	Miller, G. A. (1956) The magical number seven, plus or minus two.
numerical	... see [23] ...	[23] G. A. Miller, The magical number seven, plus or minus two. (1956)

Figure 24: Two of the main typographic styles of citation.

Although the representation (viewport) is different for the two styles, they have the same conceptual structure, which is as follows:

- A citation occurs on at least one page and may occur on many pages.

- A page does not have to have any citations, but it may have many.
- A citation must have just one entry in the reference list (a bibliographic entry).
- A bibliographic entry may relate to many citations. It must relate to at least one.

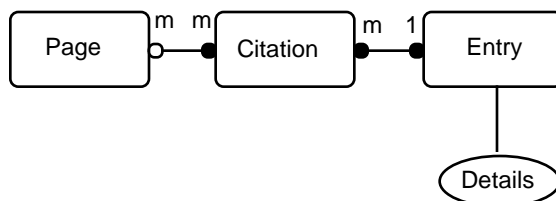Figure 25 shows an ERMIA model for this conceptual structure.

Figure 25: The conceptual structure of citations and a bibliography.

The differences between the nominal and numerical styles lie in the internal structure of the entity-store called Entry, and in the linkage between citations and entries. In the *nominal* style, the Details attribute is separated into Name, Date, and Other-details, and the combination (Name & Date) is used as the link between a citation and an entry (Figure 26). (As all academic authors know to their cost, the Name and Date mentioned in the text can sometimes fail to correspond to the Name and Date in the bibliography, so we have shown two separate entities.) In the *numerical* style, the entries are ordered by Name & Date but are then assigned a number. That number forms the link between citations and entries (Figure 27) – and here again, the cited numbers in the text may fail to correspond to the numbers in the bibliography.
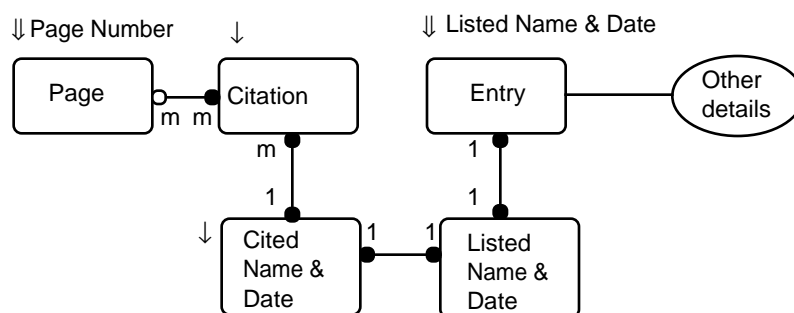

Figure 26: The structure of citations using the nominal style, as in " .. Miller (1956) ...".

⇓ Page Number   ↓   ⇓ Biblio Number

Page   m m   Citation   Entry   Other details

m   1

Name & Date

↓   1   1   1
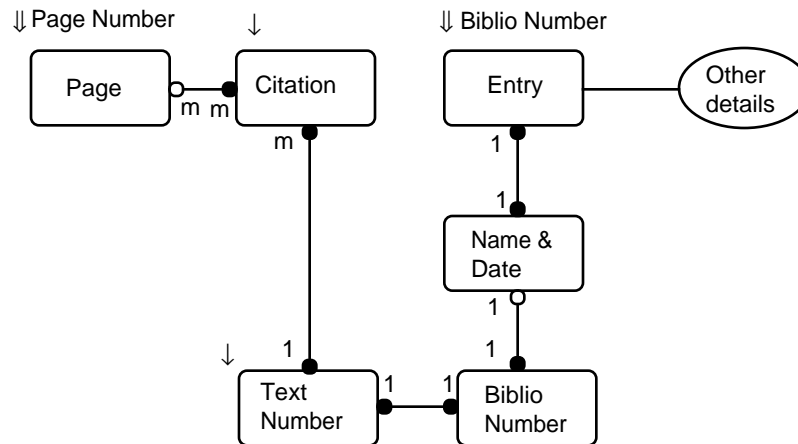
Text Number   1 1   Biblio Number

Figure 27: The structure of citations using the numerical style, as in "... see [23] ... "

The differences between forms of representation affect the cost of knowledge, because the lookup methods are somewhat different, but what we want to analyse is the cost of update, for which the task is to add a new reference; i.e. to add a new entry to the bibliography to link to a new citation in the text, and to clean up all details so that the structure is left consistent and correct. The crucial difference between the two structures is that the Name-&-Date information for an entry is unchanged by the addition of a new entry, but the bibliographic number is potentially changed (because adding a new entry changes the numbers of all entries that are below it in the bibliography).

Working with the nominal style, the method for adding a new reference is:

> note the Name and Date of the new reference
>
> search the bibliography for place to add the new Name and Date and insert the reference.

Since the bibliography is ordered by Name and Date, binary search can be used.

Working with the numerical style, the method starts straightforwardly but a 'clean-up' stage is required:

> note the Name and Date of the new reference
>
> search the bibliography for place to add the new Name and Date and insert the reference
>
> compute the new Biblio number
>
> clean up:
>
> > update all succeeding Biblio numbers in the Bibliography
> >
> > update all references to those numbers in the text.

The clean-up stage allows several methods. One is to turn the pages one by one (Page is a sorted-list entity, but it is sorted by Page-Number which has no relationship to Citation, so Page has to be treated as chain and searched serially). As each citation is found, it is updated. This method requires each page to be searched once, and needs one memory item to keep track of the current page, plus a complete mapping from old citation-number to new-citation number. (Although the change is often no more than adding 1, it can be much more complex if several changes are to be made at once, involving deletions as well as insertions.) Another method is to consider each new citation, search the pages until it is found, and then update it; the pages will have to be searched more often, but fewer memory items are needed because only one mapping from old number to new needs to kept.

The analysis illustrates how the cost-of-update can be derived from the information structure as modelled by ERMIA. A complete analysis will not be attempted in the space available.

4.4 ERMIA and GOMS compared

It is instructive to compare the ERMIA analysis with the GOMS method developed by Card et al. (1983). For this purpose we shall return to the analysis of the flat and spiral calendars by Card et al. (1994). Instead of modelling the structure of the information, they modelled the user's method of access. GOMS models are presented in considerable detail (Figure 28) which allows the authors to obtain precise predictions of timings. Unlike the ERMIA approach, in which predictions are purely symbolic and are derived from the structure model, Card et al. arrive at their prediction by considering the time taken for individual actions, which allows them to include the time taken for reading the display ('GET-YEAR') and also allows them to take into consideration such factors as the size of the target buttons for the 'POINT-TO' action.

```
GOAL: ACCESS-DAY-CALENDAR
        GET-YEAR                             ......if necessary
        GOAL: SELECT-CENTURY (1700's)
                POINT-TO (Century = 1700-1790s)
                                        => Century-display
        GET-YEAR                             ......if necessary
        GOAL: SELECT-DECADE (1710')
                POINT-TO (1710-1719)
                                        => Decade-display
        GET-YEAR                             ......if necessary
        GOAL: SELECT-YEAR (1719)
                POINT-TO (1719)
                                        => Year-display
        GET-YEAR                             ......if necessary
        GOAL: SELECT-MONTH (November)
                                        => Month-display
        GET-DAY                             ...... if necessary
        GOAL: SELECT-WEEK ([??])
                POINT-TO (23)
                                        => Week-display
        GET-DAY                             ......if necessary
        GOAL: SELECT-DAY [23]
                POINT-TO (23)
                                        => Day-display
```

Figure 28: GOMS method for locating the date 23 November, 1719 using the Spiral Calendar (omitting unnecessary details about starting the task). GET-YEAR ascertains the target year. The 'if necessary' directives allow the user to skip selecting a century if the display already happens to show the required century. GOAL sets a sub-goal which is achieved by the POINT-TO operator.

The ERMIA and GOMS approaches do not yield fundamentally different results, but they operate in fundamentally different domains. ERMIA makes explicit the information structure; GOMS makes use of the information structure, but in a purely implicit way. So the approaches are complementary, not antagonistic, and will probably be useful for different purposes and at different stages of design. We would anticipate that analyses at ERMIA's level of generality are most useful during early design, while GOMS-like analyses that are sufficiently detailed to yield close time predictions are most useful during later stages of design.

5. Conclusions

5.1 The role of structure-based formalisms

There are now many HCI formalisms. Together they form an interesting landscape which has to be viewed along several dimensions: purpose of model, style of model, and choice of fundamental entity.

Purpose of model Different modelling tools have different aims. GOMS (Card et al., 1983) takes a relatively extreme position, its aim being nothing less than accurate prediction of user times:

> "A system designer ... is choosing between having users type a key for each command and having them point to a menu ... He lists some representative tasks ... From a handbook, he culls the times for each step ... Applying the analysis from another section of the handbook, he calculates that the menu system will be faster to learn ... [and so on.]" (Card et al., 1983, p. 9)

At the other extreme are models whose purpose is solely to force the model-user to concentrate, to reflect, and to achieve a full understanding. Numerical calculations are not an issue. The algebraic models, such as PIE (Dix, 1991), have that flavour; the intention of their model is to find precise definitions for terms like WYSIWYG and 'reachability'. Duke and Harrison (1995) put this view of formal methods very clearly:

> "Although it may seem obvious, formal methods 'work' by making designers think hard about the system they are building, thus prompting questions such as 'when should feedback be available?' or 'what is the effect of this action under these conditions?' ... Any understanding of the system rests on the ease and precision with which it can be described."

ERMIA lies somewhere between these extremes, partaking of each. It yields comparative analyses of complexity for different designs, and thereby encourages reasoning about designs, but it does not lead to precise numerical predictions; the act of building a model forces reflection and (in the authors' experience) frequently leads us to new insights about underlying structure, or exposes our misconceptions, but it does not yield expressions that are mathematically as well-formed as an algebraic model. If we describe the poles of this scale as 'numerical analysis' and 'concept analysis', ERMIA lies at a point that might be labelled 'insightful analysis'.

Style of model We regard the *level of focus* as an important dimension of HCI modelling techniques. Level of focus in HCI has varied over the years. Early approaches were close-focus descriptions containing extremely fine-grain detail, whether they were attempts at cognitive modelling (such as GOMS), formal analyses of system properties (such as PIE), or natural language analyses (such as 'claims analysis', Carroll and Kellogg, 1989; Carroll and Rosson, 1991). These close-focus descriptions tended to drown the HCI analyst in a sea of detail. ERMIA is an attempt to find an effective long-focus approach, suppressing the unnecessary details.

Fundamental entity: event or structure Static devices such as paper timetables do not respond to their users and so there are no easily-observed events to model. Event-based formalisms, therefore, have only rarely been used as analysis tools for static information displays (but see the analysis of graph-reading by Lohse, 1991, using imputed mental activity such as 'find and read the X-axis label'). Structure-based formalisms, in contrast, are obviously in their element here.

Structural representations need not be based on ER diagrams, however. In data modelling, the relational model developed by Codd (1982) is a well-known alternative to ER modelling, and the parallel has been developed by Raymond and Tompa (1992) who successfully applied Codd's 'normal form' analysis to document structures. The viscosity problem, in which a writer wishing to change every occurrence of word A to word B throughout a document has to change each one individually, could be modelled as a failure to meet the required normal form, and a solution based on having a master copy of every word could be modelled as a structure that met the normal form requirements.

Although structure-based representations have the advantage that they can operate with non-interactive devices, they pay for that. Sometimes one would like to model the effects of interaction without having to adopt a new formalism. There has been some research on time-dependent ER models. Although we have not explored the possibilities the context of ERMIA, preferring as ever to favour simplicity over expressive power, Theodoulides et al. (1992) have shown that time-stamped relationships and entities can be used to model cases like limited-term driving licences which expire after a set term.

5.2 What emerges from ERMIA?

We have presented illustrations to support the following claims.

- ERMIA is not simply a sketching tool with no formality; it has a well-defined semantics coming from a long tradition of data-modelling.

- An ERMIA analysis is cheap and quick, and it can be done very early in design.

- Using this approach improves the clarity of descriptions and allows contrasts between different people's mental representations to be made explicit (e.g. Payne's folk models of ATM machines, analysed above).

- ERMIA is applicable to distributed systems, because it can distinguish the location of different information. This helps with the 'new version' problem of identifying the effects of changing part of an information system; if we know where the dependencies are, we know which ones will get broken when the system is changed.

- A symbolic evaluation of the 'cost of knowledge' and 'cost of update' can be obtained from an ERMIA analysis.

5.3 Usability issues

ERMIA does not require extensive analysis, teams of experts with stopwatches, handbooks of expected times, or any other formal apparatus. Rather, it is a classic 'back of envelope' technique. But the authors have found it does need thought and reflection, and it can be difficult to see the structure of the information at first. Practice helps, of course.

ERMIA, like its parent entity-relationship modelling, has little expressive power; many aspects of the structure cannot be modelled. For most purposes the authors have found that natural-language annotations of the diagrams are adequate substitutes. There is a trade-off between usability and expressive power, such that truly expressive systems are fearsomely difficult to learn and use. We have, if anything, erred in the opposite direction: ERMIA is very easy to learn, but may not always be expressive enough.

As a communication tool, it is too terse to stand on its own. The diagrams need to be supported by natural language commentaries. During development of a library of examples, we spent a good deal of time

talking through our models and explaining them to each other. If the diagrams are recorded without a commentary, revisiting them can occasionally be puzzling.

As an analysis tool ERMIA is well-suited to collaborative modelling, with few dependencies within the diagrams and with little need to explain intentions or look ahead. But the diagrammatic notation itself is somewhat viscous if the models are drawn using a standard drawing package. Laying them out neatly, getting everything aligned and so on takes a good deal of unnecessary work. One can envision a specialist tool that would take over some of the detail of making illustrations.

Symbolic representation is feasible, leading to executable models. The authors have implemented experimental algorithms for finding cost-of-knowledge directly from symbolic representations of information structures. The application we have developed for this purpose is little more than a proof of concept, of course, but even as such we have found it useful as a means of enforcing clarity of thought about search paths.

5.4 Where now?

Although ERMIA is not a philosophy of design, let alone a guide to good HCI practice, it has a role to play as a descriptive tool in the new approaches to HCI design now emerging. One such new approach is model-based design (e.g. Szekely *et al.*, 1993; Neches *et al.*, 1993), in which the designer creates an interface for an application by describing the application at the semantic or conceptual level. The designer then chooses the various interface components which best fit the application. Interface components include interface actions, interface objects, presentation objects representing application objects, and interaction techniques to be used with presentation and interface objects. The role of structural descriptions such as ERMIA is at the stage of describing the application at the semantic level.

Another new approach is data-centred design (Benyon, 1992, 1995), an approach that is deliberately distanced from task-based or object-oriented approaches. The claim is that data-centered approaches such as ERMIA achieve a better abstraction of the whole human-computer system.

Lastly, there are 'broad-brush' approaches, based on natural language concepts comprehensible to non-specialists, such as the 'cognitive dimensions of notations' explored by Green (1989) and DIVA (Descriptions of Interactive Visual Artefacts), developed by Tweedie (1995). In these approaches, such high-level ideas as 'viscosity' are used to characterise the properties of an artefact, and the role of ERMIA is to provide an explanation and a definition of the concepts.

In these emerging approaches ERMIA offers an approach that is comprehensible to all concerned, a notation which interface designers and usability engineers can use and which software engineers will understand. We see it as a technique which plugs a gap in existing HCI design and evaluation methods, by providing an accessible, coarse-grained analysis using a notation that is familiar to at least some members of the community and that is easily learned by all.

References

Bachman, C. W. (1969) Data structure diagrams. *DataBase*, 1(2), 4-10.

Batra, D. and Antony, S. R. (1994) Effects of data model and task characteristics on designer performance: a laboratory study. *Int. J. Human-Computer Studies,* 41(4), 481-508.

Benyon, D. (1990) *Information and Data Modelling.* Oxford: Blackwell Scientific Publications.

Benyon, D. (1992) Task analysis and systems design: the discipline of data. *Interacting with Computers,* 4(2), 246-259.

Benyon, D. (1995) A data-centred framework for user-centred design. In Nordby, K., Helmersen, P. H., Gilmore, D. J, and Arnesen, S. A. (Eds.) *Human-Computer Interaction: INTERACT-95.* London: Chapman and Hall.

Card, S. K., Moran, T. P. and Newell, A. (1983) *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum.

Card, S. K., Pirolli, P. and Mackinlay, J. D. (1994) The cost-of-knowledge characteristic function: display evaluation for direct-walk dynamic information visualizations. In Adelson, B., Dumais, S. and Olson, J. (Eds.) *CHI '94: Human Factors in Computing Systems.* New York: ACM Press.

Carroll, J. M. and Kellogg, W. (1989) Artifact as theory-nexus: hermeneutics meets theory-based design. In Bice, K. and Lewis, C. (Eds.) *Proc. CHI '89, 'Wings for the Mind'.* 69-73. New York: ACM.

Carroll, J. M. and Rosson, M. B. (1991) Deliberated evolution: stalking the View Matcher in design space. *Human-Computer Interaction*, 6(3-4), 281-318.

Carroll, J. M., Thomas, J. C. and Malhotra, A. (1980) Presentation and representation in design problem-solving. *British J. Psychology,* 71, 143-153.

Chen, P. P-S. (1976) The entity-relationship model: toward a unified view of data. *ACM Trans. on Database Systems,* 1, 9-36.

Codd, E. F. (1982) Relational database: a practical foundation for productivity. *Communictions of the ACM,* 25(2), 109-117.

Dix, A. J. (1991) *Formal Methods for Interactive Systems.* London: Academic Press.

Downs, E., Clare, P. and Coe, I. (1988) *Structured Systems Analysis and Design Method: Application and Context.* Englewood Cliffs, NJ: Prentice Hall.

Duke, D. J. and Harrison, M. D. (1995) Interaction and task requirements. *Proc. DSV-IS'95: Eurographics workshop on Design, Specification and Verification of Interactive Systems*, Toulouse, June 1995. Springer-Verlag. (in press)

Eliot, T. S. (1958) 'Whispers of Immortality'. In *T. S. Eliot: Collected Poems 1909-1935.* London: Faber and Faber.

Green, T. R. G. (1989) Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) *People and Computers V*. Cambridge University Press.

Green, T. R. G. (1990) The cognitive dimension of viscosity: a sticky problem for HCI. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.) *Human-Computer Interaction – INTERACT '90.* Elsevier.

Hartson, H. R. and Hix, D. (1989) Toward empirically derived methodologies and tools for human-computer interface development. *Int. J. Man-Machine Studies*, 31, 477-494.

Howe, D. R. (1981) *Data Analysis for Database Design.* Edward Arnold.

Hutchins, E. (1994) In search of a unit of analysis for technology use. *Human-Computer-Interaction* 9(1) 78-81.

Jacobson, J., Chistensen, M., Johnson, P. and Overgaard, G. (1993) *Object-Oriented Software Engineering.* Reading, Mass.: Addison-Wesley.

Lohse, J. (1991) A cognitive model for the perception and understanding of graphs. *Proc. CHI 91 ACM Conf. on Human Factors in Computing Systems,* pp 137-144. New York: ACM Press.

Neches, R., Foley, J., Szekely, P., Sukaviriya, P., Luo, P., Kovacevic, S., and Hudson, S. (1993) Knowledgeable development environments using shared design models. In Gray, W. D., Hefley, W. E, and Murray, D. (eds.) *Proceedings of the 1993 International Workshop on Intelligent User Interfaces.* (Orlando, Florida, January 1993). New York: ACM Press.

Payne, S. J. (1991) A descriptive study of mental models. *Behaviour and Information Technology,* 10 (1), 3-21.

Raymond, D. and Tompa, F. W. (1992) Applying normalization to notations. Technical Report, Dept. of Computer Science, University of Waterloo, Ontario, Canada.

Schwartz, S. H. (1971) Modes of representation and problem solving: well evolved is half solved. *J. Exp. Psychology,* 91, 347-350.

Szekely, P., Luo, P., and Neches, R. (1993) Beyond interface builders: model-based interface tools. *Human Factors in Computing Systems. INTERCHI '93 Conference Proceedings.* New York: ACM Press.

Theodoulides, C., Wangler, B., and Loucopoulos, P. (1992) The entity-relationship-time model. In P. Loucopoulos and R. Zicon (Eds.), *Conceptual Modelling, Databases and CASE.* New York: Wiley.

Tweedie, L. (1995) Interactive visualisation artefacts: how can abstractions inform design? In Kirby, M. A. R., Dix, A. J. and Finlay, J. E. (Eds.), *People and Computers X, Proc. HCI'95 Conference.* Cambridge University Press.

Verheijen, G. M. A. and van Bekkum, J. (1982) NIAM: an information analysis method. In T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart (Eds.) *Information Systems Design Methodologies: a comparative review.* North-Holland.