THE UNIVERSITY OF SUSSEX

AN ANALYSIS AND IMPLEMENTATION OF INFORMAL HUMAN-COMPUTER INTERACTION

Ian H. S. Cullimore

Submitted for the degree of Doctor of Philosophy

in

Cognitive and Computing Sciences

May 2000

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THE UNIVERSITY OF SUSSEX

AN ANALYSIS AND IMPLEMENTATION OF

INFORMAL HUMAN-COMPUTER INTERACTION

Submitted for the degree of Doctor of Philosophy
in
Cognitive and Computing Sciences
May 2000

Ian H. S. Cullimore

ABSTRACT

This thesis investigates the question of how cognitively informal representations can be used in Human-Computer Interaction (HCI) to facilitate the interaction between user and computer. The aspect of informality addressed by this thesis takes the form of sketch input and output, based on underlying cognitively informal representational structures.

For the purposes of this thesis an *informal interface* exhibits *tolerance* in its input and *variability* in its output, and the underlying system uses the *gist* of a representation, rather than operating on its original inherently more complex structure. The *internal representations* of informal interfaces are composed of *informal objects* that are a combination of a *prototype*, such as a straight line, and associated *informal cognitive dimensions*, such as *shakiness* and *thickness*. Informal objects can be combined into composite objects (e.g. rough straight lines can be combined to make a more complex shape such as a square). Internal representations of informal objects can be decomposed, manipulated, and recomposed, while maintaining the essential elements of the original representation.

This thesis proposes that an informal interface system may provide a useful and familiar context in which a user can work. That is, appropriate regard is given to the essential elements or gist of a representation or operation, with unnecessary clutter being removed. An informal interface system may also provide a suitable system for indexing representations, based on the addressable content of the gist of the representation. The thesis also lays out a structure for representing informal objects. Examples are given of software tools that have been developed to investigate the design of informal interfaces. The results of an evaluation of an informal interface application is also given, and further research topics and directions are proposed.

ACKNOWLEDGEMENTS

My thanks go to Dr. Mike Sharples (now *Professor* Sharples) for his limitless enthusiasm and energy for this project, Yvonne Rogers and Steve Easterbrook for their valuable contributions and support, the excellent COGS department at the University of Sussex, and everyone else who has helped me along the way.

DECLARATION

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other University for a degree.

Signature:

_____

Ian H. S. Cullimore

1. Informal Interfaces

1.1 An Overview of this Chapter

This chapter introduces the concept of informality in human-computer interaction, and discusses how such concepts are already being used in everyday life. Both Human-Computer Interaction (e.g. user interface design) and the underlying representational data structures are considered. Consideration is given as to why it might be useful to use informality in computing, and the potential benefits and drawbacks.

An overview is given of the key concepts of informal interfaces, for which definitions and examples are given. These key concepts are explored in greater detail later in the thesis, in Chapter 3. A number of examples are given of how informality might be used to good effect in the design of software programs. There is also a discussion of the pros and cons of using informality in interface and software design.

Finally, an overview is given for the methodology adopted in conducting the research for this thesis, and the structure of the thesis is laid out in a chapter by chapter summary.

1.2 Introduction - Informality in Everyday Life

As computers become integrated into everyday life, we interact with them in more informal ways than we used to. The latest microwave ovens, for instance, can now be asked to cook food until they perceive that the food is ready, instead of having to be set to cook for a pre-set number of minutes and seconds. Modern cameras are informal in the sense that it is possible to just pick them, point and shoot; they do most of the work of focusing and calculating the

required exposure, rather than the user needing formal knowledge about the technology of photography such as shutter speeds, aperture settings and so on. Products like the Apple Newton Personal Digital Assistant (PDA) and the KidPix children's drawing package are informal in their level of interaction with the end user, through the use of handwriting input, gestures, and the use of sketching. These are examples of a continuing trend away from command-driven interaction with the computer, towards co-operative systems (Robertson, Zachary & Black, 1990). In such systems, the user indicates high-level intentions and constraints, and the computer manages low-level operations and supports the user in framing new intentions.

Such instances of informality are becoming more common, and there has been research into some aspects of it, such as using sketches in collaborative design (Scrivener & Clark, 1994). This thesis continues this line of research to address the issues of how notions of informality can be infused into the design of computer interfaces and the internal operations of computer systems, what internal representational structures could be suitable, and what the potential benefits and drawbacks might be of such systems. This approach addresses the issue that, while some of the research that has been undertaken is notable for its use of sketch in input and output, it has tended to rely on a superficial front-end sketch interface (Meyer, 1996), with the design of the underlying internal representations based on conventional software engineering ideology.

It has been known from the times of Leonardo da Vinci (Fish & Scrivener, 1990) that sketching is an effective aid in allowing the mind to run freely. However, there has been little analysis performed of how to effectively use sketching both *externally* (as a medium for input

and output) as well as *internally* (in the sense of an underlying internal representation being used to store and manipulate the overlying informal representations). Figure 1 shows a drawing of "a house in the sun" by a five year old. This is recognisable to almost all adults from the same cultural background, and children of at least that age, as just that. The widest scope of the question "why is this image recognisable as a house?" is one addressed by many different areas of research, such as cognition and vision. This thesis concentrates on the question of how the essential elements of a representation (its *"gist"*) that can be used effectively in the design of computer interfaces, and the underlying software and representation structures.

Figure 1: Drawing of "a house in the sun" by a five-year-old (context: white, male, Anglo-Saxon, British citizen, residing in England) 1997

This thesis sets out to present a cohesive structure for a particular type of informal interface, to detail the research and findings, and to describe a methodology for investigating and constructing such an informal interface system. This thesis does not attempt to address the

wider issues of informal interfaces in general, but addresses one type of informal interface – that of a personal computer based system, utilising informality in the form of graphical sketch input and output.

The thesis investigates how a particular type of informal interface could facilitate an informal and more familiar mode of interaction between a user and a computer. Furthermore, the internal representations used in such an informal interface may sometimes provide a useful structure for data representation, one that concentrates on the essence of a meaning or representational state rather than a set of more formal parameters.

1.2.1 Formal and Informal – Dictionary Definitions

Let us consider some dictionary definitions for "formal" and "informal". The Concise Oxford Dictionary defines "***formal***" as "*used or done or held in accordance with rules, conventions or ceremony*", "*precise or symmetrical*", "*perfunctory, having the form without the spirit*", and "*of or concerned with (outward) form or appearance, esp. as distinct from content or matter*".

As for "***informal***", the dictionary provides the definitions "*without ceremony or formality*", and "*everyday, normal*".

So by these definitions we might judge that an "informal interface" would be one that goes against conventional rules, is not necessarily precise, and in some way captures the spirit of what the user is trying to do.

1.2.2 The Important Issue of Context-Dependency

Note that it is necessary to consider the processes of human cognition, since meaning is *context dependent*. What is meaningful in one context may be irrelevant, or hold a different meaning, in another. For instance, an informal suspension bridge designers' software package would embed different constructs of meanings for structures, constraints, attachments and so forth from an informal garden design program. There can therefore probably be no entirely general informal interface system, but rather different systems adapted for differing domains. Also, within a particular informal interface system, regard must always still be given to context, the purpose of the system, the level of detail required, and so forth.

1.3 Informality in Computing and Interfaces

"Informality" is by nature a wide-ranging and sometimes vague term. This section discusses the type of informality addressed by the thesis.

Relaxing the constraints of formality in different systems is an interesting exercise. This will mean different things in different types of formal systems. For instance, a system may be "formal" through its formalised architecture of data structures. Or, it may seem "formal" to a user through the rigidity of operation. A system may be deemed to be 'formal" by users through sticking to conventions of usage.

Applying a relaxation of formality, and thus increasing the level of informality, may not make sense in all systems. For instance, it may not be clear as to what would be meant by "informal" data structures. However, according to this thesis informality can be applied to input and output interactions in computer systems. Conventional input systems are formal in one sense in

that the user is constrained as to what interaction choices are available. Physically, keyboards (with a finite number of keypress combinations) and mice are only a physical interface to an underlying presentation interface such as a GUI (graphical user interface) windowing system.

This thesis concentrates on the application of informality to user interface design – that of sketch-like input and output on a conventional computer graphics display. Here, *sketchiness* is the application of informality. Using sketch for input and output is "informal" in the sense that it is sometimes a familiar and easy way for humans to interact, and is often well suited to natural, creative processes. A user may sometimes feel more at ease interacting with a computer through informal and familiar sketch rather than using a more formal and conventional system of mouse, keyboard and monitor. Sketch is also informal in the terms of this thesis because the underlying representation (or gist) of the substance of the sketch is tolerant of the imprecision of the input data and output presentation. Such a seemingly superficial front-end to a user interface exhibits interesting consequences both for the internal representations used in the system, and also for the style and perception of operation by end users. For instance, imagine the scenario that someone has been requested to send directions to a colleague as to how to find an office for a meeting. If the two were in a room together, one likely way this interaction could take place would be for one to sketch out a rough map for the other on a piece of paper. The map would show, in freehand style, the essential elements of roads of importance, intersections, turns to be taken, and so forth. Such a solution is difficult if not impossible in the scenario that the two workers are distant from each other, and required to use a computer (e.g. by using a text-based email system) to effect this communication. Alternative ways of accomplishing this task would be to (a) sketch a map on a piece of paper and fax it, (b) sketch a map on a piece of paper, scan in the image to a

computer using a document scanner, attach the graphical binary image (e.g. a jpeg file) to an email, and send that to the colleague, (c) use a drawing package such as Visio (Microsoft, 2000) to create an electronic image of the map using the conventional tools of mouse, menu selections etc, and the email this file, and so forth.

Now, there are two things about the very first way of accomplishing this task, i.e. sketching the map on the piece of paper, which are essentially informal. One is the actual mode of interaction – familiar pen-on-paper, with imprecise hand-drawn lines. It does not matter too much how good the rendition is, as long as it gets the essential meaning (the "gist") across. And that is the second essential element of an informal interface according to this thesis – the underlying gist or meaning of the representation. Again, it does not matter within rough terms exactly what information was input, as long as the essential meaning of the representation remains the same. So precisely how the straight lines for the roads were drawn does not matter. They could have been drawn in slightly different ways, by different hands, and still retain the same meaning or gist in the overall context of the map.

1.3.1 Definitions of Informality and Informal Interfaces

By "informality" in this thesis we mean a relaxation in the way that a user interacts with a computer, and a relaxation in the way that a computer retains its internal states and data representations. That is, users usually interact with conventional desktop computers through the use of a mouse, keyboard and monitor. There are formal sequences for interacting with the computer and the software programs running on it. For instance, a user has a specific series of steps to go through to create a word processing document, enter text, print the document out, and so forth. Similarly, to create a diagram of a map showing how to get to a particular place

(perhaps to someone's office), it is usual to go through the formal steps of running an appropriate software package, entering lines and other drawing objects using the mouse and menu selections, and so forth. A more informal way of accomplishing this might be for the user to be able to sketch the diagram directly into the computer free-hand using a stylus, or to be able to manipulate operations of the computer through gestures, etc.

By an "informal interface" in this thesis we mean one that is tolerant of imprecision in input protocol and output presentation. Furthermore, an informal interface might use underlying representations of data, state and so forth that are more to do with the essence of the meaning of the state or data. For instance, consider the example of creating a diagram of a map. A formal method and interface would result in a formal data structure of, say, an object-structured rendition of the map. An informal method and interface would result in an abstract representation of the map that contains all the necessary and pertinent data, but might not necessarily produce exactly the same map when reconstructed. Nevertheless, in the same way that two sketches of a map by the same person would be unlikely to be exactly the same bitmap image but would mean exactly the same thing to this person, so the reproduced image would still convey *exactly* the same cognitive information.

Note that while the interface itself is informal and relaxed in nature, the actual *structures* of the internal representations used may be formal. That is, the objects of the interface and underlying representations are informal in the sense that they have cognitive interpretations of informality (such as "gist"). However, the data structures used to represent these informal objects are formal in nature. For instance, in programming terms they may be constructed from C++ objects, or frames, or some other formal data representation method.

1.4 Key Concepts

By using the term "informal" in user interface design and interaction we are referring to interfaces that are *tolerant* of the user's input (the user has flexibility in choice of action) and that show *variability* in their output. In an informal interaction there is a many-to-one mapping between an input event (e.g. a menu selection) and a state change in the notional machine, and a one-to-many mapping between the state of the notional machine and an output presentation.

For example, many examples of simple hand-drawn straight lines map to the one notional representation of what this thesis terms *an informal straight line*, and this single representation in turn would reproduce many examples of reproduced rough (i.e. looking like hand-drawn) straight lines.

The aim of this thesis is to propose a framework for defining cognitively informal interaction between a user and computer. By *interaction* we mean the process flow between user and computer, which in turn is defined by the *external* and *internal representations* of the interaction. By *external representations* we mean such things as images displayed on a computer screen (whether as input by a user or as displayed to a user), and by *internal representations* we mean the way in which the computer stores the information describing such objects in a form to be operated upon, displayed, transmitted or the like. Consider the example of a pull-down menu system; here the user is constrained to a finite set of possibilities of function choices, each of which maps onto one state, and each such state is shown as a single or finite number of presentation choices by the interface. Conversely, an informal interface may map a possibly infinite number of different input events onto a single state of the notional machine, with the state being the *gist* of the interaction. By *gist* we mean a

representation of the essence of the meaning of a state, as discussed further in the next section. Each state of the notional machine may be presented at the interface in a variety of forms, governed by the constraints of the internal representation and the restrictions of the output device.

Thus, this thesis researches the idea of applying notions of informality to both the user interface and also the underlying representational and operational structures. Such informality might be construed as being somewhat superficial, if only applied to the immediate front end of the user interface, but such concepts can also be applied at a deeper representational level.

1.4.1 Further Definitions of Terms – Tolerance, Variability and Gist

There are three key concepts behind informality in user interface design - *tolerance* in input, *variability* in output, and the *gist* of a representation - corresponding to the three stages of input state, output state and the internal state of the system. By *tolerance* we mean allowable differences in input function mapping to a singular internal representational state. By *variability* we mean that an internal representational state can be mapped in a number of ways to an output mechanism without appearing to have a different meaning.

By *gist* we mean a representation of the essence of the meaning of a state. That is, the simplest and highest level of abstraction of a state when its attributes of tolerance or variability are absent. For instance, the gist of a rough hand-drawn straight line is simply the *concept* of a *straight line*. This may be hard to identify and measure, but it can be done by, for instance, ascertaining that the essence of the meaning of the representational state, as judged by its input and output states (which will be different), is judged to still be the same. Furthermore (in a

given context), a number of objects having different visual appearances, but judged by humans to me the same meaning, will be mapped by an informal interface system onto a single internal representation state. Similarly, such an internal representation state will in turn be mapped to visually different (sketch-like) output, judged by humans to have the same meaning.

For instance, imagine a rough straight line, hand-drawn on a piece of paper by a human. If the human were asked to draw a number of examples of a rough straight line (perhaps on separate pieces of paper) then it is unlikely that any of them would be *exactly* the same if analysed in minute detail. However, to the human who created them, each rough straight line would in one sense hold exactly the same meaning – *a straight line*. So in this example the *gist* of the input representation is a *straight line*. The *tolerance* of input means that the exact way the representation is entered does not necessarily matter – all of the similar rough straight lines mean *straight line*. If the collection of these images, on separate pieces of paper, were to be shown to a number of different other humans, then we would expect them all to decide that the drawings meant *straight line*. So the *variability* of output does not have a notable effect – the essential gist of the representation has been conveyed successfully. Also, the character of tolerance or variability does not affect the gist. For even though some of the rough straight lines might also be described as "a wobbly line" or "a diagonal line", taking away any elements of the associated properties of the image (e.g. wigglyness) would leave the bare essential element of the gist of the image, i.e. a straight line.

Thus through the application of input tolerance there are an (infinite) number of ways to input a representational state into a computer system through an informal interface, all of which are deemed to be equivalent representations. Similarly, through the application of output variability

there are also an (infinite) number of ways of representing a computer state, all of which are also deemed to be equivalent representations.

There may also be a (possibly infinite) number of ways of representing the gist of the state internally, but each implementation will use only a single internal representation for each gist. That is, there are many possible informal interface formalisms, but only one would be used in each particular implementation.

1.4.2 Levels of Representational Equivalence

There are three levels of *representational equivalence* according to this thesis:

1) Two representations are *truly equivalent* when there is no measurable difference between them

2) Two representations are *cognitively equivalent* when there is a physical difference between them, but they appear the same to the user

3) Two representations are *cognitively informally equivalent* (“*CI-equivalent*”) when there is a physical difference between them which can be perceived by a user, but they have the same meaning still to the user.

As an example, consider the case of sketches of a house, as portrayed in Figure 2.

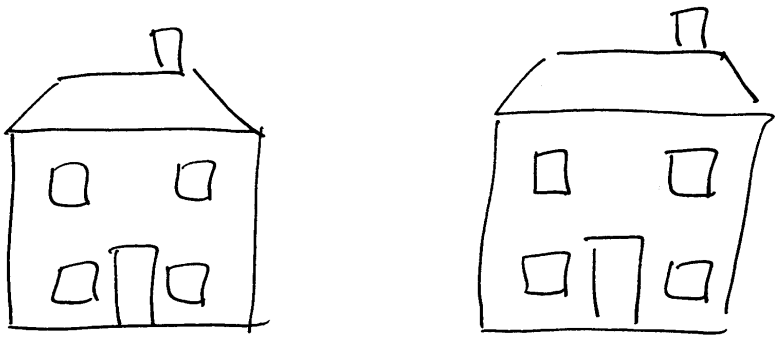

Figure 2: Two sketches of a house

If the two sketches were an exact pixel bit-map copy of each other, e.g. by means of a photocopy or a cut-and-paste operation, then one is a true equivalent of the other.

Two slightly different versions of a sketch of a house, viewed separately at different times by a user and deemed to have been the same sketch, would be cognitively equivalent. In this case the user is not aware that the two versions are actually slightly different – to the user they appear the same.

Two slightly different versions of a sketch of a house viewed together at the same time (or even at different times) by a user would be CI-equivalent if they were deemed to represent to all intents and purposes the same original image (or gist) in a particular context. In this case the user is aware that the two versions are slightly different, but has decided that this does not matter (in a cognitive sense) in this particular context – they have the same meaning.

1.4.3 An Example of Informal Interfaces and Interactions

As an example, suppose we had a user interface which had been constructed on the basis of pen-based sketch and gesture input, and similarly sketchy (albeit computer generated) output on a monitor or LCD panel. The application could be, perhaps, a video whiteboard system, with two such units connected across an Internet TCP/IP connection. Such two devices could be separated by miles, or even on other sides of the world.

Suppose the user on one end wanted to convey the rough idea of a new bridge design that they had been working on. A simple video whiteboard system would allow the user to input, with the use of a stylus on a touch-sensitive panel, a pixel bit-mapped graphical representation of the bridge, which could be transmitted (real time) to the distant user.

Such as system could, using conventional recognition and object systems, resolve the sketch into a collection of intrinsic recognised shapes. These could then be conveyed to the distant user, and reconstructed in some manner to provide a similar rendition of the original sketch.

Likewise, an informal video whiteboard system could be contrived. The user would sketch in the initial design, and the informal system would resolve the rough sketch into its intrinsic informal components of lower-level objects such as straight lines and rectangles, and if applicable it would build these up into higher-level constructs such as box girders, supports, struts, and so forth.

Here, there is a distinction between a sketch recognition to object system (a formal system, in the terms of this thesis), and the proposed informal system. The formal system handles object recognition along conventional lines, according to the literature. The informal system similarly

handles object recognition in some manner, but with the fundamental construct of tolerance of input, variance of output (perhaps deliberate), and resolving the sketch objects to their gist as the fundamental representation.

Thus although a single user, or a number of different users, might input slightly different pixel bit maps, the system would map each input sketch to the same internal representation for the same class of bridge. This is an example of the *tolerance* of input. For instance, a number of people might sketch in a number of types of bridge. These could, for example, be recognised and resolved into their fundamental type, such as a suspension bridge, or a drawbridge. So if a number of people produced sketches of a suspension bridge, the system would resolve them all down to his fundamental type, or their gist. As another example, a sketch of the Golden Gate Bridge in the San Francisco Bay Area would not be much good if the bridge were to be represented as a box girder type. On the other hand, few people would know the exact ratio of, say, the span of the Golden Gate Bridge to the height of its support towers. It would be possible to produce many sketches of the bridge that would be convincing renditions to most people, even though the actual dimensions may not be exact.

What would then be transmitted across the wire, rather than a stream of, say, pixel co-ordinates, would be tokens of the internal representations (the lines, boxes, struts and so forth), their relationships, and a representation of the nature of the informality. This is an example of the *gist* of the representation.

These tokens would then, in turn, be reproduced on the distant user's screen.

There is at this point a choice of exactly *how* to map the representation to an output image (the *variability*). This could be a "formal" depiction, similar to an engineering drawing using straight lines, perfect curves, and so forth. However, a creator might not regard this as cognitively informally equivalent to the input information, because it represents a different meaning. For example, an engineering drawing might be read as meaning box girder of length 3 metres, whereas the creator meant box girder of undefined length.

The variability of output is a crucial part of the operation of an informal interface as described in this thesis, because the fact of this variability reinforces what is the intended meaning of the representation. If an output machine (perhaps a computer program) were to be instructed to redraw the image, then each example would be (deliberately) slightly different. However, each slightly different image would still convey exactly the same gist (in a correctly working system, of course), and users presented with multiple representations should be able to separate the gist of the representations from their incidental features.

The computer system should therefore depict the representation in its own rough sketch-like rendition, either based on its own internal depiction engine algorithms, or perhaps mimicking the user's own style. In this way, while there is *variability* of output, the gist remains a consistent mapping function across the communication mechanism.

1.5 Measuring Success in an Informal Interface

Note that a fundamental principal of the effectiveness of an informal interface system is that the resultant gist or intent of the relayed image is cognitively indistinguishable from the gist of the original rendition. That is, the images are *CI-equivalent*. For instance, a rough sketch of a

house on recomposition might display all the inherent *crucial characteristics* of the original design, such as the number of doors and windows, and the rough position of the chimney. Note that this is context dependent. A rough sketch of a house to be presented to a glazier might dwell more on the detail of the designs of the windows, and panes of glass. A sketch to be shown to an architect might be more concerned with the overall essential elements of the type of roof (perhaps pitched), and the number of chimneys.

This thesis proposes two ways of measuring CI-equivalence in informal interfaces: a) user appreciation studies, and b) a feedback loop.

1.5.1 User Appreciation Studies

In user appreciation studies, one could set up a focus group of a number of people, to whom original and reconstructed images are shown under suitable control. A successful set of informal representations of images would be one in which a large majority of people decided that the images were CI-equivalent. Once again such a definition is rather loose and informal, the metric this time of the degree of CI-equivalence being the level of "user satisfaction". Note that there are two separate parameters under consideration: both the *level of informality*, and *CI-equivalence*. Note how the boundary conditions apply. If the degree of informality is zero (i.e. a formal system), that is there is never any variability between image and the resultant reconstructed image, then the user rating of CI-equivalence (actually just cognitive equivalence in this case) should be at a maximum since the resultant image is *exactly* the same as the original. On the other hand, if the degree of informality is so high that the rating of CI-equivalence is zero, then the representations are all so distorted as to convince the users that none of them are supposed to be the same image.

As the degree of informality increases, so user satisfaction may change. However, it is proposed that in an informal interface system as defined in this thesis, user satisfaction can be a maximum even with a non-zero degree of informality. For as long as the images are CI-equivalent (although not *actually* identical in a physical sense of rendition), then there is no loss of gist to the users.

However, at some point as the degree of informality increases, so too does the degree of *malformation* of the resultant image - and hence user satisfaction falls or completely fails.

1.5.2 Feedback Loop

In the feedback loop system, the resultant output image is fed back in again to the informal interface system as a new input, and hence around the loop again.

By definition, a system exhibiting *CI-equivalence* will create a state that is an equivalent meaning to its input, which when fed back in as input will produce equivalent output. In such a CI-equivalent system the transformation function is actually an identity function *I*, so no matter how many times around the loop the image goes it will always be recognised.

Of course there may (and indeed will) be variability in the output of the image, as that is the definition of an informal interface. But this is one of the crucial parts of an informal interface system; even though the output image varies, it signifies the same state. Indeed, the deliberate variability of output is an important aspect of such informal interfaces. For instance, a number of examples of an output image of a sketched house might all have slightly different pixel bit maps or vector traces on a computer graphics display, but the success of the informal interface in retaining the gist represented, is that to all intents and purposes they represent the same

notion - i.e. a house of a particular style and set of essential attributes. In this case, the essential attributes might be the number of windows, number of chimneys, whether the porch has pillars, and so forth. So it would be necessary to have different CI-equivalent interfaces for different uses – that is such systems are domain and context dependent.

Note again that such informal representations are context and domain dependent. For instance, an estate agent, architect and young child would probably create quite different renditions of a sketch of the same house. Also, sketches of the same house would probably be different depending on whether they were to be used for selling the house, designing it, using it in a cartoon, or whatever.

So a measure of the success of an informal interface system is a pass or fail of the loopback test. A successful informal interface system should always be a CI-equivalent identity function, and so the loop will iterate indefinitely. A failure during some iteration of the loop means that the informal interface is not a CI-equivalent identity function, and the gist would not be recoverable on input.

1.6 Informality Already?

Some systems already display some of properties of informal interfaces. For instance, the architecture of HyperText Markup Language (HTML), commonly used in World Wide Web sites, allows for variability in output - it is up to the designer of the Web browser (for instance NCSA Mosaic, Netscape Navigator or Microsoft Explorer) to decide exactly how text and images are laid out, and how controls such as buttons might be displayed.

The relatively new programming language Java, too, has informal elements. Its platform *independence* actually causes its presentation style to be platform *dependent*, as with HTML, and hence open to interpretation and variability of output. Platform independence and execution on a virtual machine (the "Java VM") can lead to variation in implementers' interpretation of some modes of operation, such as the exact functioning of widgets such as buttons or edit boxes.

1.7 So Why Informal Interfaces?

The term "informality" in everyday usage suggests a lack of precision, an easing of social or linguistic conventions, and in the context of this thesis the use of sketch to represent rough ideas and interactions. The benefits of informality include being able to express a vague or partially understood idea, and being able to explore the essence of a concept without being committed to its eventual form. The sketch serves as a framework on which the mind can build. An informal interface is an analogy of the sketch in human-computer interaction, relaxing the conventional input/output constraints of current user interfaces, in order to offer the user a more relaxed environment.

1.7.1 Informal Graphs

Consider the case of a spreadsheet. A request to display a graph of the relationship between elements of the data may result in an, albeit graphically pleasing, over complex and crowded visual picture which may not bring out the essential elements of important data relationships. By displaying the output in an informal, sketchy way, showing only the important *gist* of the information, the user may be alerted to crucial information stemming from deeper underlying

relationships while not being distracted by the detailed nature of the displayed information. So the gist of the information might be an underlying trend in the way that a function varies. But the deliberate roughness of the rendition of the graph (variability of output) would mean that the user is not able to read too much detail into, say, the exact point at which two lines cross.

The potential benefit of a particular type of informal interface systems might manifest itself in a number of ways. For instance, in a suitable context, informal sketch-like input and output methods (e.g. the user using pen input on a pad to convey input information, and the computer producing graphical sketch-like output on a screen) may make it easier for the user to both communicate desires and information to the computer, and for the computer to present the correct level of detail. It may be the case that by presenting too much information to the user the essential gist of the information does not come out.

Figure 3 represents the example of an informal graph versus a formal one.

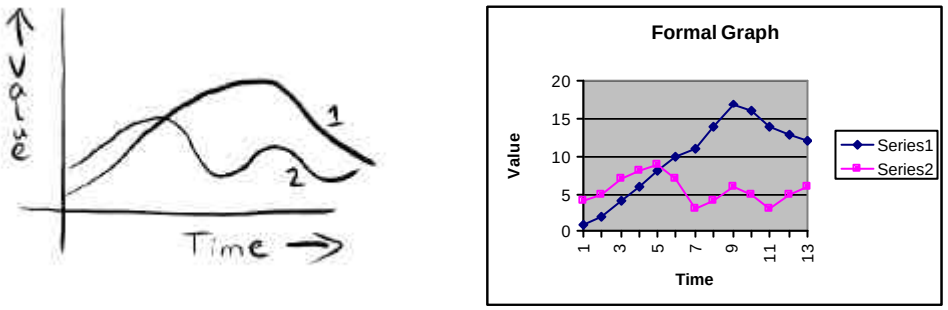

Figure 3: An Informal Graph Versus a Formal One

1.8 Pros and Cons: Informality versus Uniformity and Stability

It is not claimed that the application of informality will always be the best way of designing user interfaces and their underlying representations. There are certainly cases where informality (or

one particular aspect of it, perhaps the variability of output) is undesirable. For instance, in the example of the design of a bridge above, all users of the bridge when it had been constructed would hope that it had been built from detailed drawings and plans which had been subjected to rigorous testing and analysis.

On the other hand, in the early stages of the example of the design of a bridge as described in section 1.4.3, it may be more helpful for designers to be able to more freely and quickly play around with their incomplete designs, while homing in on their preferred construction given the limitations and constraints of, perhaps, the surroundings or terrain.

Applying informality to interface design and layout may sometimes have its drawbacks, from both a user perspective and also from the viewpoint of commercial considerations. From a user perspective, in order to construct and maintain a mental model of the functioning of a program (Norman, 1986), a degree of uniformity and stability in the underlying structure of the program is required.

1.9 A Methodology for Research into Informal Interfaces – An Overview

The thesis takes an heuristic approach to investigating some of the concepts and boundaries of applying informality to interface design. It might be thought at first that informality could be applied only to the design of the upper-level interface layer itself. However, there is also benefit in applying similar techniques to the lower internal representational layer.

An initial issue to be considered is how to generate realistic looking sketch objects, such as the fundamental *Rough Straight Line* (*"RSL"*). Experimentation with different software algorithms can show the effectiveness or otherwise of differing ways of representing convincing lines; the

metric of success being how "human-like" a resulting straight line looked to the human eye. By varying relevant parameters of the algorithm, and sometimes discarding seemingly irrelevant or conflicting ones, lines that are convincing to a greater or lesser extent can be created. `This process eventually leads to a seemingly successful and useful representational structure for an informal straight line, with the potential of being suitable for representations of other informal objects.

For instance, it is found useful (that is, the result is effective and realistic RSLs) to utilise *informal dimensions* such as *shakiness* and *period*. Other attributes such as *harmony* prove to be of less value. These informal dimensions are described in detail in section 3.4.

Multiple instances of such primitive objects can be combined into more complex ones; for instance, multiple straight lines can be the basis of more complex structures such as squares, rectangles, triangles, or grids. This leads to the need for extra parameters such as *constraints* (Leler, 1988) and in particular *attachment*. In this context attachment is a simple form of constraint, whereby for instance one particular end of an informal rough straight line is constrained by being attached to one end (or perhaps a more descriptive informal constraint such us "the middle", which may not be *exactly* the middle) of another rough straight line. As an example, in Figure 4 (1) line B is attached to line A in the middle (or, in an informal sense, it is attached *roughly* in the middle), at (roughly) right angles. So, if line A is rotated by a certain amount to the position as in (2), then it must follow (through the propagation of constraints) that line B ends up still attached to the mid-point of A, and (roughly) at right angles. Applying constraints removes the need to *directly* apply the operation to the constrained object – i.e. there is no need to apply the operation of rotation to line B.

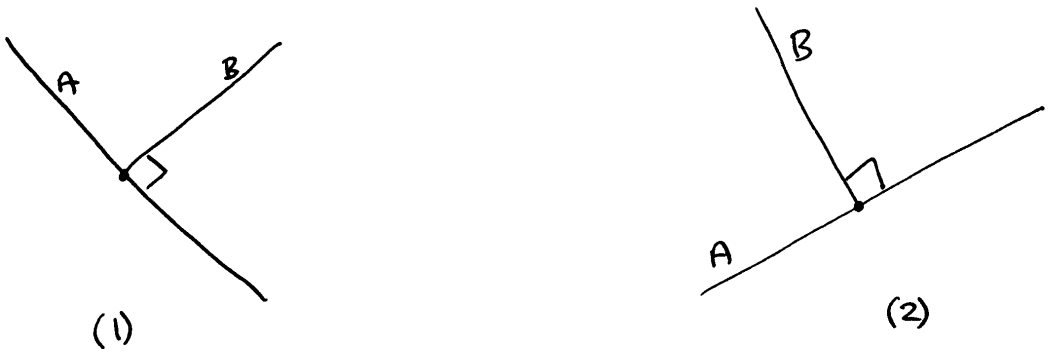

Figure 4: Example of constraints

With this methodology it is possible to build up a knowledge of how to create representations of informal objects in an effective manner, and to identify a set of parameters in the form of informal cognitive dimensions on which to base the representations. With this in place, we can establish an architecture for representing structures of such informal representations, thus providing a foundation for implementing this type of informality in user interface design.

1.10 Structure of the Thesis

Having laid out an overview of the current state of user interface designs, their shortcomings, and an opening analysis of the introduction of informality into such designs, this thesis aims to lay out a cohesive rationale and architecture for informal interface design. First is a detailed review of the literature, in Chapter 2. This encompasses not only more mainstream Computer Science reports mapping the history, evolution and possible future directions of interface design in HCI, but also (as is necessary in a multi-disciplinary subject such as HCI) relevant elements taken from areas of research such as Cognitive Science, Cognitive Psychology, and

Art and Design. The later sections of the chapter bring an up to date account of what other activity there has been in recent times in informal interfaces. The thesis then goes on in Chapter 3 to examine in detail an architecture for informal representational structures; the aim here is to show that an informal interface can be much more than just a superficial "informal-looking" sketchy front-end to a conventional rear-end formal operating environment and system (although this is still one viable and useful instantiation). Rather, by extending the notions of informality down into the fundamental building blocks (e.g. the internal representations) of a system, the concept can be retained throughout in both a stand-alone *single node* implementation (that is, a singular system whereby a single user interacts with a single computer), and also a *multiple node* system involving a web or network of many users interacting directly with their computers, and indirectly with the computers and users within that network.

After this, in Chapter 4 the implementation work that has been carried out is examined in some detail. The thesis describes a number of software programs, developed in a range of environments depending on the scope of the system module such as 'C', C++ and Prolog (for an AI "intelligence engine"). The platforms used are PC compatibles, using operating systems such as DOS, Microsoft Windows 3.11, Windows 95, 98 and ME, and Windows NT. The code developed is entirely applicable to other platforms such as the Apple Macintosh or Unix systems, and given suitable development tools the code should be portable to a wide variety of platforms. The later sections of the chapter detail some user reaction studies.

Having laid out an overview, the literature survey, designs and implementations, representations and case studies, Chapter 5 summarises the overall scope of the work, draws conclusions, and discusses future research directions.

Finally, the appendices lay out the detail of much of the essence of the work. Full or partial listings are given of the software developed, examples are given of intermediate format representations, and a syntax for representational structures is presented.

1.11 Summary of this Chapter

This chapter has discussed how informality in computing is starting to occur more frequently in everyday life, but how there has been little research in applying such notions of informality to human-computer interaction and the underlying software representations and operations. It has examined the advantages of sometimes having greater informality in interface and software design, when sketch input and output can be used to advantage to allow a user to interact with concepts at an appropriate level of specification, rather than being required to specify meanings that are not currently part of the users intentions.

The chapter has set out the key concepts behind informality in interface design, and has defined commonly used terms. It has given some examples of how informal interface software might be used in everyday life, and how the users might benefit from such designs.

The chapter has also discussed both the benefits and potential drawbacks of allowing informality into user interface design, but has argued the case for greater use of informality - in the correct context - to allow for new types of applications and user interface experiences.

The chapter then set out a methodology for conducting research into informal interfaces, and described the subsequent chapters in this thesis in outline structure.

2. From Leonardo to AARON - a Review of the Literature

2.1 An Overview of this Chapter

The study of any area of Human-Computer Interaction encompasses a necessarily wide range of disciplines, such as computer and cognitive sciences, psychology, formal methods, art and design, and philosophy. This chapter surveys related work which is of relevance to informal interface design in a number of areas: (1) user interface design methodologies, theory and practice, (2) the emerging use of sketches in graphical interfaces, (3) principles of representation theory, and (4) design and development tools and methodologies.

Although this area of synthesis of HCI research necessarily brings a number of disciplines under one umbrella, the aim of the thesis is to concentrate only on those aspects of existing literature and research which are most pertinent to informal interface theory and practice. Many of the fundamental topics of informal interfaces are explored in great detail in other disciplines, such as sketching in art and object recognition in image processing. This thesis concentrates on using the basic principles from other disciplines where necessary, and seeks to not replicate other research directions but instead to concentrate on the fundamental objective of introducing informality through the use of informal sketch input and output, and underlying informal representations, into the computer science of user interface design and implementation.

Section 2.2 introduces an historical perspective of the evolution of Human-Computer Interaction. Section 2.3 presents an HCI perspective on a route towards informal interfaces: section 2.4 discusses suitable representation frameworks, and section 2.5 summarises the

issues involved in choosing development methodologies and tools in the implementation of informal interface software in this thesis. Section 2.6 examines the emergence of the metaphor of Sketch in HCI, with section 2.7 detailing more recent events in the emergence of informality. Finally, Section 2.8 summarises the main points contained in this chapter.

2.2 Introduction – Informality in Interface Design

Informality can be characterised in a number of ways, and this thesis explores just one of those characterisations - namely tolerance of input and variability of output exemplified in the application of "human" sketch-like input and output presentations with an underlying structure of informal representation. Other conceptions of "informality" are possible. For instance, some might regard systems that are "human-like", "multi-modal", "malleable" and "usable without training" as informal. It is not our intention to constrain the application of the term to computer interfaces. However, it is important to recognise that the term "informal interface" is used in a very specific way in this thesis.

While we are not claiming that introducing informality is necessarily a better way of designing interfaces, or mandating the complete re-architecture of interfaces along informal lines, we are claiming that conventional (more "formal") interface systems and their underlying formal representations are sometimes lacking in their modes of functional operation. Some of these deficiencies can be addressed and remedied through the application of notions of informality to the interface, interaction modes, and the whole "user experience".

Consider the history of user interface design since the conception of widely-used computer systems. Early mainframe computers, with their "dumb terminals", allowed only slow character

"teletype" (TTY) keyboard and monitor communication between user and computer. Primitive graphic displays were later introduced. This was still the prevalent situation when desktop "Personal Computers" with greatly increased local processing powers were introduced to the world, starting with machines like the Altair, then progressing to the now ubiquitous IBM PC and its compatibles.

In these systems the user was highly constrained in the flow of interaction. Input and output is very formal: the user had a single method of input, a keyboard. The user was highly constrained as to how to communicate with the computer, such as having to use an arcane set of commands (e.g. Unix's "*ls -al*" or "*rd*") the syntax of which had to be followed exactly to avoid a breakdown in communication (Banahan & Rutter, 1982).

Similarly, output to the user was tightly constrained; not only in the physical medium (an arbitrary rectangular grid of an equally arbitrary number of lines and columns of textual characters or bit maps), owing its design principles to electromechanical, rather than human factors constraints, but also in the syntax of the command language output. The user was continually having to adapt his or her *modus operandi* to that of the computer. Of course, "human-oriented" presentation is not necessarily the same as "informal" presentation, but informality in presentation is one way that human-oriented output can be achieved, and may be effective when used correctly.

Progress was made with the advent of Graphical User Interfaces (GUIs) following the initial work of the Star development team at Xerox's PARC (Johnson, Roberts, Verplank, Smith, Irby, Beard & Mackey, 1989) principles later adopted by product developments at, for instance, Apple with the Lisa and Macintosh computers, and Microsoft with the Windows

operating system. However, from an informal interface perspective not much has changed; output is certainly in a more visual form. Typically, though, input is through the conventional keyboard, with an extra dimension being added by the mouse. But on closer inspection the user is still constrained by a finite set of commands (the "pull-down" menu system), and has to accept whatever presentation system is on offer. The interface is a veneer on top of a conventional operating system architecture; in the case of Microsoft Windows this is literally so, since it is a protected mode graphical environment built on top of a 16-bit real mode operating system with its design from a previous decade.

2.2.1 Prior Developments of Influence

Sketchpad (Sutherland, 1963) and ThingLab (Borning, 1979) have been an influence on this thesis. Both are graphical interface systems allowing users a high degree of expressiveness and control in their inputs, and have strong underlying operation engines. They show the developing path of graphical interfaces, object-oriented design and development, and the notion of using constraints. In the terms of this thesis they both exhibit some tolerance of input, although little variability of output. But they were instrumental in spurring on the thesis' central theme of using sketch as an input mechanism, and indeed extending this to deliberately maintaining the output also in a sketch-like form. This leads to an analysis of the requirements for the underlying representational structures, which while loosely based on Sketchpad and ThingLab (in their use of constraints and objects) enhances their constructs through the realisation of the need for the addition of informal types of *cognitive dimensions*, as well as *prototypes*.

Ivan Sutherland's (ibid.) seminal work on developing what was truly the world's first GUI, Sketchpad, at MIT's Lincoln Laboratory on the TX-2 computer provided the basis for much

of modern-day HCI. His system, based on a lightpen, a bank of switches and a CRT display, was the first interactive computer graphics interface. Interestingly, although "sketch" by name, Sketchpad was not particularly "sketchy" by nature. Although a hand-held lightpen was used for input, the system was really designed to produce highly precise drawings with perfect lines and corners. It also pioneered the use of constraints on the geometrical objects.

Alan Borning's ThingLab (ibid.) is a constraint-based simulation system developed in the object-oriented programming language Smalltalk. ThingLab, heavily influenced by Sketchpad, is a graphical system designed to allow a user (a programmer, really) to set up operators and constraints between objects, and to simulate simple physical experiments. ThingLab does not have any embedded knowledge of any particular domains, but provides tools and mechanisms for creating applications.

2.3 Informal Interfaces and HCI - The Traditional Literature

The discipline of Human-Computer Interaction is now well established in the fields of Cognitive and Computer Science, Psychology, Ergonomics, and other areas of research and development. This multi-disciplinary subject draws in researchers, implementers and users from all walks of life and work, and is the subject of a wide range of study. Its basic concepts are now well understood and documented, and there are many publications on the principles of HCI (e.g. Preece, Rogers, Sharp, Benyon, Holland & Carey, 1994). In this book, Preece et al. discuss the now "conventional" aspects of HCI: cognitive frameworks, perception and representation, mental models, interface metaphors, input, output, interaction styles, and design methods and techniques. They (ibid.) state that "… HCI is about designing computer systems that support people so that they can carry out their activities productively and safely…". They

point out that Donald Norman (Norman, 1988) identifies two key principles that help ensure good HCI: *visibility* (controls need to be visible and have good mapping of their effects) and *affordance* (controls should suggest, i.e. afford, their functionality).

Preece et al. (ibid.) highlight a divergence in the approach to research on either side of the Atlantic in the 1970s and 1980s. They point out that American pioneers were more concerned with how computers could enrich lives, make them easier, and facilitate creativity and problem solving. European researchers, meanwhile, were constructing theories about HCI such as *usability*, and the development of operational criteria and assessment metrics.

They (ibid.) define a list of disciplines contributing to HCI: (a) computer science, (b) cognitive psychology, (c) social & organisational psychology, (d) ergonomics & human factors, (e) artificial intelligence, (f) linguistics, (g) philosophy, (h) sociology, (i) anthropology, (j) and engineering & design. They go on to highlight the fact that *graphical representation* is the main method used in conveying information at the interface, and that *mental models* are important to people when interacting with devices and systems. Studies have been carried out (e.g. Rogers et al., 1992) to investigate this, and the general assumption is that people do use some type of model, but that it is often incomplete and vague. This is an important issue for informal interfaces in HCI, where the *tolerance of input* and *variability of output* may be used deliberately to try to more closely coincide with a user's (possibly vague) mental model of the system. Conceptual models are important for informal interfaces too. The design model, the user's model and the system image (Norman, 1986) is a classic architecture for a conceptual model. Here, the internal representational structure in the system image of the

informal interface system is deliberately "informal", to avoid introducing meanings that do not match those intended by the user.

Preece et al. (ibid.) also include a discussion on input devices, the classic ones being keyboards and mice. They go on to discuss pen input devices, but only in the context of handwriting recognition and gesture recognition. An informal interface, as described in this thesis, uses pen input at a fundamental level: as a natural stylus device conveying sketch input. Output devices are the other half of the input/output equation. They (ibid.) discuss the conventional output devices used in HCI: Graphical User Interfaces (GUIs), sound, virtual reality and multimedia. The informal interface (as described in this thesis) uses conventional graphical output, but in a form that is designed to avoid conveying information not intended in the object represented.

The authors (ibid.) discuss on interaction styles, such as *command entry*, *menus and navigation systems*, *natural language dialogue*, and *direct manipulation*. Informal interfaces as presented in this thesis do not assume a new and unique form of interaction style: conventional ones (principally menus) are used, although there is scope too for the use of direct manipulation.

Other such works, e.g. Carroll (1991) and Thimbleby (1990) also set out the basic principles for HCI: the fundamental cycle of *input*, *output*, and internal *models* and *representations*.

The human factors of graphical human-computer interfaces have been analysed by Maguire (1985). Designers have a wide array of input and output devices and facilities available. This

subject is of relevance to informal interfaces as presented in this thesis, as similar input techniques, especially inking, are utilised. In this paper, drawing techniques are discussed:

1. *point plotting* (drawing straight lines by specifying start and end points)

2. *polygon filling* (filling an enclosed polygon by specifying any point within it)

3. *erasure*

4. *autoplotting* or *inking* (leaving a trail of 'electronic ink' like a pen; freehand sketch can be accomplished in this way)

5. *rubber-banding* (dynamically stretching a straight line from a start point to an end point)

6. *grid snap* (connecting all inputs to the nearest point of a background grid, thus enabling precise drawings to be created more easily)

7. *libraries of symbols and figures.*

Interestingly, Preece et al. (ibid.) contains a discussion on *formal versus informal groups*. "An increasing number of researchers believe that informal, spontaneous, communication is as important as formal communication, if not more important…". However, this use of informality is not directly relevant here. The use of "informal" sketching described in this thesis relates to the representations employed in communication rather than the protocol of communication. The authors (ibid.) also consider, in envisioning design, the use of *sketching*. They (ibid.) state that "sketching techniques can be useful for exploring all kinds of design ideas…" and suggest the use of visual brainstorming (Verplank, 1989) to explore alternative designs. Clearly, this stresses the value of paper and pencil as a rapid means of producing designs. However, it is

relevant to this thesis in the sense that the ideas operated through this kind of process are likely to be at a general level of specification and systems should not force the user to specify meaning other than that intended. Informal interfaces offer an approach to this problem.

Dix et al. (Dix, Finlay, Abowd & Beale, 1993) discuss emerging technologies such as multi-sensory systems, speech, handwriting and gesture recognition, and animation. Their introduction to handwriting recognition is mainly concerned with character recognition in pen-based systems. But they go on to state "if we were to design an organizer from scratch we may well decide to do away with the keyboard … we can consider all sorts of other ways to interact with the system … we may decide to use drawings to tell the system what to do … the different input device … opens up a whole host of alternative interface designs and different possibilities for interaction". This is a stimulus for the selection of the informal interfaces developed here, i.e. an interface system primarily based on using sketch for input and output. In other words, the interface would have wide application.

This thesis aims to build on these foundations for conventional HCI, and to utilise the existing methods and ideas. So, the conventional framework of HCI design, such as the adoption of the stages of *interaction tasks*, *psychological aspects*, *design*, and *evaluation*, is utilised (Preece & Keller, 1990).

The use of sketch recognition has been the subject of study in the field of computer graphics interaction techniques. For example, Foley et al. (Foley, Wallace & Chan, 1984) describe a mechanism for using a sketch recognition system to determine commands that are entered by using a device such as a mouse, a tablet or a light pen in a sketch-like manner.

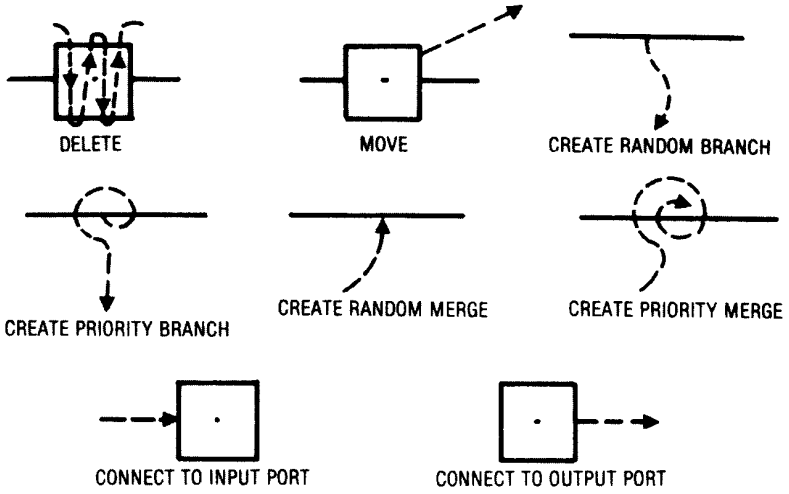

Figure 5: From Foley, Wallace & Chan

In this way *gestures* can be sketched in to provide one way of having command and control over a computer system, as illustrated in Figure 5. Strangely perhaps, they (ibid.) use formal diagrams (straight lines etc.) to illustrate their sketched gestures, instead of actual examples of sketched gestures. The relevance here to the thesis is the use of an informal means of interaction – sketch – as a way of providing input to a computer system.

Although not directly relevant to this thesis, there are a number of instances in the literature where researchers have investigated the nature of other forms of informal communication and interaction between users. This is of interest in the context of this thesis, as it helps to set a framework within which the research presented here can sit.

Fish, Kraut and Root (1992) performed an evaluation of video as a technology for informal communication. They state "Collaborations in organizations thrive on communication that is informal because informal communication is frequent, interactive and expressive…". The authors' focus is on their audio/video technology system, *CRUISER*. They conclude that "some form of desktop videoconferencing could prove useful in preserving informal communication channels for geographically distributed organizations".

Hollan and Stornetta (1992) also identify the need for informal interaction. They state that there is a fall-off in likelihood of collaboration between researchers as distance increases. They go on to state this occurs "because of the large number of informal interactions necessary to create and maintain working relationships".

Researchers at the Xerox Palo Alto Research Center (Elrod et al., 1992) have developed an interactive video whiteboard system called *Liveboard*. This system provides for stylus-based group interaction with a computer-supported whiteboard system. Users can interact with, control, and annotate (with sketch and handwriting) conventionally displayed computer graphics. A Liveboard application *Tivoli* is later used to examine informal workgroup meetings (Pederson et al., 1993). This is further explored in Moran et al. (1998). In this paper the authors develop the idea of using a freeform electronic whiteboard metaphor to enable users to interact using pen-based scribbling and editing.

Researchers at Hewlett-Packard's Bristol research laboratory (Whittaker, Frohlich & Daly-Jones, 1994) studied informal communications in the workplace. By this they mean it to be "… brief, unplanned, and frequent…", supporting a number of different functions: the execution of work-related tasks, co-ordination of group activity, transmission of office culture,

and social functions such as team building. The authors studied methods for (often remote) informal communication such as video and audio systems, "glance" systems, and roaming conversations. However, they did not study sketch (perhaps on a whiteboard or a napkin) as a means of informal communication. They conclude with a note on the "sheer *brevity*" of informal communication. They state that this may be because, in the case of formal communication, if participants are uncertain that they may meet frequently then they may condense multiple issues into a single interaction. However, in the case of informal communication, familiar interactants know that future conversations are guaranteed.

Zhao investigates gesture-based diagram editing (Zhao, 1993), giving some insights into the recognition of hand-sketched diagrams. Most other approaches use either gesture recognition or visual language parsers, whereas this author proposes an incremental paradigm of gesture recognition and a co-operative communication for pattern recognition and diagram parsing. He (ibid.) identifies the need for both a Low Level Recogniser (LLR) and a High Level Recogniser (HLR) - the LLR acting at a single stroke level, and the HLR transforming these basic symbol sets into editing commands. In the context of this thesis, this is important as it later helps to set out a framework for an informal interface system.

2.3.1 Informal Interfaces and Computer Vision

This thesis does not presume to cover the classic subjects of shape recognition and the interpretation of line drawings from Computer Vision (Marr, 1982 and Marill, 1989). Computer Vision is more concerned with the problems associated with recognising lines, shapes and relationships (e.g. *behind, in front of*) from a photographic or other bit-map image. These scenes usually come about in a different fashion from those found in the type of

informal interface system described in this thesis. In computer vision, the scene is presented to a recognition engine as a *fait accompli*, typically as the output from a video camera or scanned photographic image. The problem then is to extract the embedded data (in 2-D or 3-D) from the underlying clues of lines, curves, shades and so forth (Leclerc & Fischler, 1992). On the other hand, informal interface theory as presented in this thesis is more concerned with the analysis (in real-time or at post-input time) of sketch-like data input by a user, its internal representation and potential interpretation, and the corresponding utilisation of sketch-like output to convey the appropriate level of informality to a user. However, the literature of Computer Vision does provide useful algorithmic and representational concepts relevant to informal interface systems, such as best-fit schemes for straight lines (Pao, Li & Jayakumar, 1992, and Chattopadhyay & Das, 1991), and shape recognition (Marr, 1982).

Informal interfaces have one advantage over computer vision: the input mechanism accepting line drawing data from the user is able to receive (and possibly process) that data in real time, which leads to associated extra data which can be inferred. For instance, a line drawing of an object such as a *house* will have been drawn in a particular sequence: perhaps first the outside lines of the rectangular body of the house, then the windows and doors, then the roof, and finally the chimney. This temporal data may help in resolving ambiguous images. The lines comprising the sides of a Necker Cube will have been drawn in a particular order, which may (arguably) be used to interpret the desired resultant image. Reisberg (1987) explores this in his paper on "External Representations and the Advantages of Externalizing One's Thoughts", in which he argues the importance of externalising one's thoughts, i.e. "…by sketching the content of mental images…"

Negroponte (1971 & 1973) defined sketch recognition as "the step by step resolution of the mismatch between the user's intentions (of which he himself may not be aware) and his graphical articulations. In a design context, the convergence to a match between the meaning and the graphical statement of that meaning is complicated by continually changing intentions that result from the user's viewing his own graphical statements." His HUNCH program was one of the early attempts at using inferences from a sketch to end up with a final design.

Note that an informal interface system, as described in this thesis, faces many of the problems of computer vision. Some of these problems might continue to be intractable. For example, it may never be possible to develop systems that are able to interpret graphical input in a way that exactly matches what the user intended.

Citrin and Gross (1996) describe a technique for the recognition of diagrams, whereby low-level and high-level components are utilised. The authors detail a method for pen-based input and diagram recognition using a PDA and desktop computer. The PDA performs low-level shape recognition and the desktop performs high-level recognition. This is an architecture similar to that employed by Zhao (1993), and the one used in the implementation of an informal interface described in this thesis.

As Straforini et al. (1992) state, "the recovery of the 3-D structure and the recognition of viewed objects from TV images are among the main goals of computer vision…". They describe a system in which a low-level vision module recovers line drawings from real images, and a high-level reasoning module to further process the image. Marill (1989) examines the problem of how the human vision system produces three-dimensional interpretations of two-dimensional images. Other shape recognition techniques, for instance using the Straight Line

Hough Transform (Pao, Li & Jayakumar, 1990), also exist. The Hough Transform is a well-known method for the detection of parametric curves in binary images.

2.4 Some Representational Frameworks

This thesis proposes a structure for informal interfaces in Chapter 3. However, the key concepts behind a representational architecture are reviewed first here.

Fundamental to the representation proposed is the concept of a *primitive prototype* and associated *specific parameters*. As an example, consider a rough hand-drawn straight line. In this case, the gist of the representation is a simple *straight line*. The primitive prototype of the representation is the abstract geometrical perfect straight line of no thickness that passes through the locus of points according to a particular best-fit algorithm. The associated parameters are a particular set of attributes that further adequately describe the actual line, e.g. a measure of the shakiness of the line, its colour, and so forth. This viewpoint is derived from two main sources: the concept of *frames* (Minsky, 1975) from Knowledge Representation theory, and parameters in the form of Cognitive Dimensions (Green, 1989).

2.4.1 Frames

The conceptual structure of frames suits the prototype/parameter construction well. A frame is a structure that represents knowledge about a limited domain, and is basically constructed of a fixed *prototype* and a number of associated *slots* that are occupied by *fillers*. The use of frames has been applied to a number of problem areas of Artificial Intelligence, such as representing knowledge for a class of recognition problems (Kuipers, 1975). Frames have also provided some use in programming languages, the Apple Newton PDA being one

computer with a development environment programming language based on frames. Frames have also been instrumental in the development of object-oriented architectures and programming techniques, used heavily in the software implementations in the thesis.

2.4.2 Cognitive Dimensions

Green proposed the notion of "cognitive dimensions" (Green, 1989) as a descriptive vocabulary to more accurately describe relevant interface qualities in cognitive rather than computational terms. He introduces notions such as *viscosity* (a measure of how resistant representations are to change) and *premature commitment* (a measure of whether a user becomes fixed to one option too early in the interaction process), and explores these concepts in a series of papers (Green, 1990, 1991a and 1991b).

Such cognitive dimensions provide a basis for developing the notion of attached (cognitive) parameters. Cognitive dimensions in general describe aspects of information structures, so that these aspects of an interface can be described in the same way as describing an object in terms of physical dimensions (such as weight or length). According to Wood (1992), Green originally proposed cognitive dimensions as a set of concepts for characterising different computer programming languages. However, the concept of cognitive dimensions lends itself well to describing interfaces.

2.4.2.1 In Defence of Using Cognitive Dimensions in Informal Interfaces

The concept of cognitive dimensions is taken up in this thesis and later used as a basis for fillers for the slots of prototypes (in the style of frames), in the guise of *informal cognitive dimensions* such as *shakiness, thickness* and *period*. These informal cognitive dimensions

are not cognitive dimensions in the original style and sense of Green, as they are varying in nature and necessarily of different types. Nevertheless, Green's use of cognitive dimensions is a stimulus for the use of similar informal counterparts, as used in this thesis.

2.5 Some Software Tools

Before going on to examine some of the threads of development that have been influential in forming the basis for this thesis, it is instructional to review some of the background tools and technology available to aid in constructing representational structures and to help develop demonstration software.

Although the architecture of frames (Minsky, 1975) outlined in section 2.4.1 provides for an appealing basis for internal representations, there are few software development tools available using the concept. To some degree frames have lead on to the more generalised concept of o*bject-oriented analysis and design* (Booch, 1994), as used in some of the software implementation in this thesis using C++ (Borland, 1992). Other software in the thesis has been developed in the classic 'C' programming language (Kernighan & Ritchie, 1978).

Of note is the use of the "Artificial Intelligence" programming language, Prolog (Clocksin & Mellish, 1981), to develop some of the core "intelligence engine" software (Amzi, 1995). Prolog is well suited to the problems encountered by the recogniser engine of the software implementation described later, in section 4.4.6. Here, the problem faced is to derive higher-level constructs (such as "triangle", "square" and "house") from sets of lower-level primitives (e.g. rough straight lines). The intelligence engine is supplied with raw data in the form of a prototype (a line) and attached parameters (such as length, start point, and so forth). Prolog's

backtracking and cuts are well suited to this type of problem – deriving the mid-level constructs of *attachment* (actually a constraint), and from these the high-level constructs of primitive geometrical objects (e.g. a triangle is three lines A, B and C, with A attached to B, B to C, and C to A).

2.5.1 Constraints and Constraint Programming Languages

It will be seen later (in Chapter 3) that part of the structure for an internal representation of informal interfaces relies on an explicit or implicit use of *constraints* (e.g. Borning & Duisberg, 1986). For example, if four lines are drawn in the shape of a square then they are constrained by the fact that one joins to the next at its end point, and that next line is joined at its end point on to the next one, and so forth.

Figure 6 gives an example of a sketch-like square generated by one of the software programs implemented in this thesis (called "i-Fax"). Its internal representation is composed of the informal objects of four rough straight lines, structurally constrained through the *attachment* of one line to another.

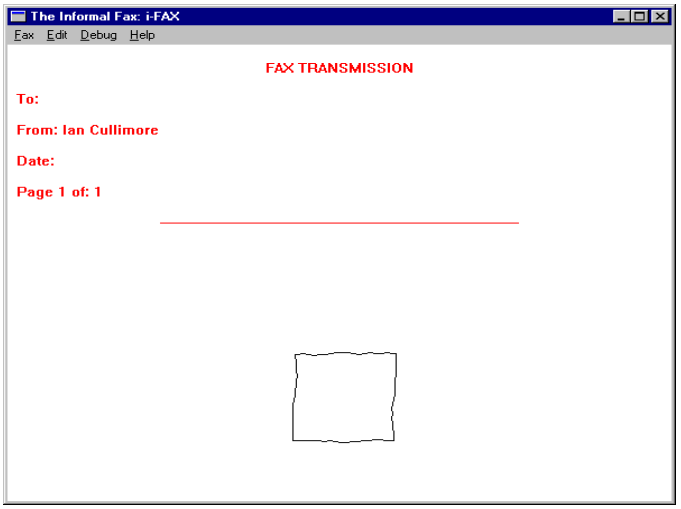

Figure 6: A sketch-like square generated by i-Fax

There are two types of constraints utilised in the informal interface system described in this thesis: *global* and *structural*. Global constraints are informal cognitive dimensions that are applied uniformly to objects throughout their lifetime, such as shakiness, period, thickness, length, and direction. This system uses one structural constraint, *attachment*. Other systems might utilise other structural constraints such as *above*, *to_the_right_of*, and so forth.

Constraints are powerful in that if this square were to be rotated, there are a number of ways that this could be accomplished. For instance, matrix manipulation could be applied to the pixel bit-map array that represents the screen display. However, if constraints are applied then *just one* of the lines needs to be transformed to a new position – the rest by their attachment must follow suit. (Indeed, *attachment* is the primary constraint utilised in the software developed for this thesis).

The concept of constraints is an important one for the type of informality in interface construction described by this thesis, as it lends itself naturally to being a useful relationship to bind primitive graphical elements (such as *rough straight lines*) together.

This concept was used much earlier in Ivan Sutherland's seminal work (Sutherland, 1963) on the constraint-based graphical interactive system Sketchpad, and in Alan Borning's ThingLab (Borning, 1979), as later expanded by Borning & Duisberg (1986) and others such as Stefik (1981). There has been continuing work in the field of constraint systems, such as multi-way (as opposed to one-way) constraints in the DeltaBlue algorithm (Sannella, Maloney, Freeman-Benson & Borning, 1992), constraint-based dataflow (Kass, 1992), and CONSAT, a system for constraint satisfaction (Gusgen, 1989).

2.6 The Power of the Sketch

There has, at the same time, been an increasing awareness of the power and usage of diagrams and *sketching* in related disciplines. Lansdown (1985) points out that computer graphics designers tend to aim at photographic realism when "convincing naturalism" might be more appropriate. Bundy (1977) demonstrates a need for diagrams to describe problems in the mechanical world, such as whether a block sliding down a slope (the "roller coaster" problem) will reach the top of the other side, or loop the loop. His solution is to describe aspects of the problem from the diagrams as symbolic descriptions, to be passed to a general problem solver for the mechanical world.

As Cohn, Randell & Cui (1993) state in their work on qualitative spatial relationships, the development of ontologies for spatial logics based on regions has only recently started to become a serious research activity. They describe their work of refining a system for qualitative reasoning, based on the relationships between elements of diagrams of systems and object such as INSIDE, OUTSIDE, JUST_OUTSIDE and so forth. Although their examples are based on reasonably well drawn diagrams, it is interesting to note that rough sketches would also serve the same purpose in most cases.

An increasing amount of research is being conducted in the use and understanding of diagrams, both in general and in special cases such as *graphs*. Researchers such as Preece (1983) and Lohse (1991) have examined the issues behind understanding graphs. As Lohse states, despite the increasing importance of graphics in the design of information systems, there is only a partial understanding of how people perceive and process graphic information. He indicates how research into cognitive models for the perception and understanding of graphs can be

applied to informalism and how rough-sketch representations of graphs can be inherently interesting as informal renditions of information. One of the key concepts from informal interfaces is the relaxation of the invariance of output by computers, so it is revealing to study how people perceive and process meaning from graphical information, although this is not the primary focus of this thesis. However, the understanding of graphs is one example of an application in the future of some of the elements of the type of informal interface system described in this thesis. Lohse describes a computer program UCIE (Understanding Cognitive Information Engineering), which models the underlying perceptual and cognitive processes used by people to decode information from a graph, and considers results from the analyses of bar charts, line graphs and tables. His mechanistic approach is to first determine a logical sequence of eye fixations that will be able to decode the information, and then calculate (from known observations about Short Term Memory, reading times, and so forth) how long this will take.

Lewis, Mateas, Palmiter & Lynch (1996) provide an example of the potential power of a sketch-like graph, if used in the appropriate context. Their paper presents a process for using ethnographic data to drive design in a product development environment. Part of their process involves collecting data through analysis meetings. Interestingly, they illustrate the temporal structure of an observational data analysis meeting in their paper through the use of a sketch-like graph, shown in Figure 7. As the authors (ibid.) state, "(the figure) is not meant to indicate the precise temporal structure but rather to provide an example of a 'typical' analysis cycle in such a meeting". So the authors have not tried to represent the information that they are trying to get across to the reader my means of a formal, perfectly drawn, graph. Instead, they opted to convey a rough impression of what they were trying to show by means of a more sketchy,

rough-drawn, and informal graph. In this way, the reader should be able get a feeling for the general gist of their point. However, the reader should also realise that it is not appropriate to read too much in to the precise elements of the graph. It would probably not be appropriate, for instance, to interpolate or extrapolate the line data to a high degree of precision.

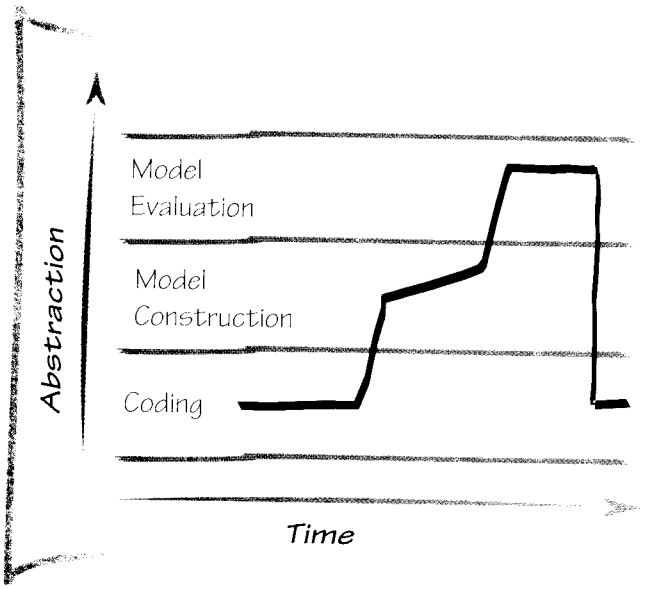

Figure 7: An example of a rough sketch-like graph from Lewis, Mateus, Palmiter & Lynch (1996)

There is a large body of work on using stylus input with computers. For instance, the National Physical Laboratory illustrate the need for sketch input and subsequent analysis in their Electronic Paper project (Brocklehurst, 1989). In this, a user can write on a flat panel display, the handwritten symbols, drawings, characters and script are interpreted, and the "intended" result is displayed on the screen. NPL Electronic Paper presents one primary characteristic of an informal interface: *tolerance* of input. The user is able to input data in the form of sketches and gestures, which the software then "cleans up" for subsequent display. For instance, two rough axes for a graph could be input by a user, and their end points annotated. These would then be tidied up by the software and displayed as perfect straight line axes, with intermediate

annotation tick marks along the axes. Then the user can enter data points, and the "correct" type of curve selected (e.g. linear, quadratic or whatever). This curve is then calculated and drawn by the program.

There are other examples of the continuing research into the use and understanding of graphics and diagrams, such as the examination of the intersection of computer vision and computer graphics (Montalvo, 1985), and the acquisition and validation of qualitative visual properties (Montalvo, 1990). In this paper, a "knowledge visualiser" software program represents graphical objects, properties and relations as *frames*. Properties, which are represented on a computer graphics screen as *prototypes*, can be "incrementally combined to form more complex properties and objects". Each property, represented by a frame, has a slot for the property itself, as well as a *generator*, *recogniser* and *echo* function. So for instance, for the property of SIDEDNESS, a triangle would fill the slot with the value 3, as illustrated in Figure 8. This is of relevance to the construction of an informal interface as described in this thesis, as it is an example of an internal representational structure based on a frame-style *prototype* and associated *fillers* for the *slots*, along the lines proposed in Chapter 3.

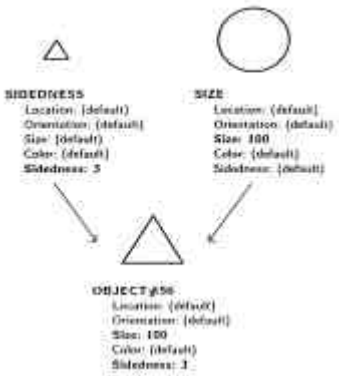

Figure 8: From Montalvo (1990)

Thinking with diagrams, and the emergence of visual programming languages, is a continuing theme of the use and power of graphics in traditional domains. Green and Blackwell (1996) cite some of the reasons behind this, such as the fact that people find it easier to deal with the concrete (i.e. visual representation) rather than the abstract, that human cognition is optimised for vision, (i.e. shapes are easier to process than words), and visual programming makes semantic relationships explicit (in the form of a picture). The subject of thinking with diagrams is relevant to informal interfaces as described in this thesis because of the association of diagrams with informal, familiar, sketched renditions.

Another hugely influential figure in guiding the synthesis of art, sketch and computers has been Harold Cohen (McCorduck, 1990). Although an accomplished artist by training and vocation, and knowing little if anything about computers (at a time when they were still in their infancy), Cohen virtually threw away his established career to pursue a vision of creating art through computers. His computer program AARON, "the only program currently in existence capable of the autonomous generation of works of art" (Sharples, Hogg, Hutchison, Torrance & Young, 1989), is capable of creating sketch-like drawings of artistic scenes.

Figure 9 shows such a drawing created by AARON. From a computer science perspective AARON used not to be, in a sense, well founded architecturally. It is true that AARON is generally considered to create original works of art, judged by most to be aesthetically pleasing and possibly indistinguishable from work that might have been created by a human. On the other hand, Cohen is not a computer software engineer by trade or training (and would probably never claim to be such, either), and in the early days of AARON's development the

computer program was a jumble of hard-coded software. Later development has put AARON's software on a more structured and frame-oriented basis.

A discussion of AARON is relevant to this thesis because it is an example of internal representational structures used to depict human-like graphical output. It is also an example of the use of variability of output, as AARON somewhat unpredictably produces similar although not identical pictures of the same theme. So the overall gist of a scene might be the same (e.g. a group of people standing amongst trees), although the precise detail might not.

Figure 9: Harold Cohen, 1986 (McCorduck, 1990, p 6)

In art and design there has been much work done in analysing the principles behind sketching. Fish and Scrivener (1990) state "Leonardo da Vinci advocated the use of untidy indeterminacies for working out compositions, because he believed that sketches stimulated visual invention". Also, according to Fish and Scrivener, "…sketches are incomplete visual structures that amplify the inventive and problem-solving uses of mental imagery…", and that Negroponte (1977) noted that "sketch recognition is as much a metaphor as fact. It is illustrative of an interest in those areas of design marked by vagary, inconsistency and ambiguity. While these characteristics are the anathema of algorithms, they are the essence of design". Although the authors were talking about sketch in a somewhat different way, it is a stimulus to the research in this thesis. Informal interfaces as described in this thesis are inherently sketch-like in nature. Further work has been made in the area of Computer Supported Co-operative Work and Collaborative Design. Scrivener and Clark (1994) argue that the sketch, far from being "… anachronistic … a technology out-performed and perhaps to be superseded by computer based imaging technology", in fact is still a vital function because "… the characteristics of the sketch are ones that support and facilitate the kind of visual reasoning engaged in the early stages of design, as does the actual activity of sketching".

These authors, along with others (Scrivener, Harris, Clarke, Rockoff, & Smyth, 1993, and Scrivener, Clarke, S., Clarke, A., Connolly, Palmén, Smyth, & Schappo, 1994), furthermore argue the case for dispersed work group design efforts being supported through the use of interactive real-time sketching functionality. In such a system, designers separated by large distances (and possibly even located in different continents and in different time zones) would use a shared computer "sketchpad" system connected through telephone lines.

Barfield, van Burgsteden, Lanfermeijer, Mulder, Ossewold, Rijken and Wegner (1994) describe some examples of sketch icons, in an article on interaction design at the Utrecht School of Arts. In this article, much use is made throughout of sketch instead of more formal diagrams and tables, as illustrated in Figure 10 and Figure 11. This is a continuing trend in using such informal modes of presentation where once formal methods would have been chosen. This may be in part to do with technology (the ease with which sketches can now be combined into word processed documents, for instance), and in part to the authors' (in this case notably a collection of Art School designers) desire to convey a particular impression of content style.

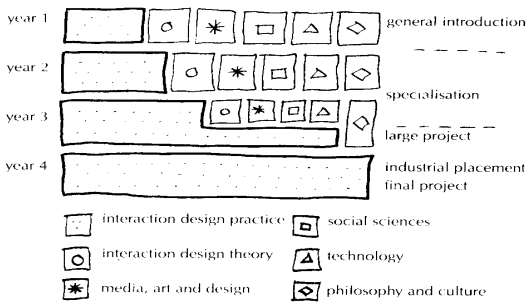

Figure 10: Sketch used in a paper on Interaction Design, from Barfield et al. (1994)

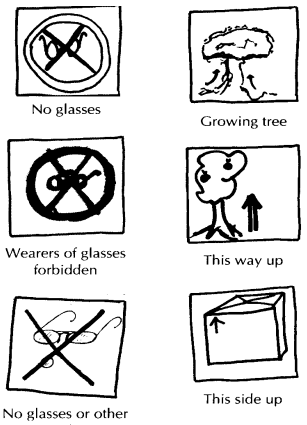

Figure 11: Sketch-like Icons (Barfield et al. 1994)

Citrin and Gross (1996) have been working in the College of Architecture and Planning at the University of Colorado on the concept of distributed digital sketchbooks, using a front-end Personal Digital Assistant (PDA) hand-held computer (an Apple Newton), and a companion back-end computer (an Apple Mackintosh). The back-end computer provides support to the PDA in the form of higher processing power and greater storage. Here the need is to, for example, assist telecommunications service engineers in the field. The authors explain that typically a worker arrives at a site to repair a unit, only to be confronted by a confusing tangle of wires; the worker's initial task is usually to make a sketch diagram of the mess to try to sort out exactly what to do. The authors' PDA system allows the worker to input these sketch diagrams, for a first-pass recognition attempt, and then subsequent up-load to the back-end conventional computer for further processing. In this way these early, and immensely useful, sketches are not wasted (because the same need would arise to make yet another sketch the next time an engineer is called out), but instead saved, edited and otherwise modified for later download.

The back-end to the system is their (ibid.) Electronic Cocktail Napkin project (Gross & Do, 1996a), as depicted in Figure 12. In this paper, the authors argue: "in all design domains (from software to submarines), diagrams and sketches play a key role in the conceptual, formative stages. We want to support this drawing and sketching, the kind you might do on the back of an envelope or a cocktail napkin. It is quick and rough, but it lets you explore and explain basic alternatives quickly".

In a further paper (Gross & Do, 1996b), the authors state "interfaces for conceptual and creative design should recognize and interpret drawings. They should also capture users'

intended ambiguity, vagueness, and imprecision… Freehand drawing can provide this information and it is a natural input mode for design…"".

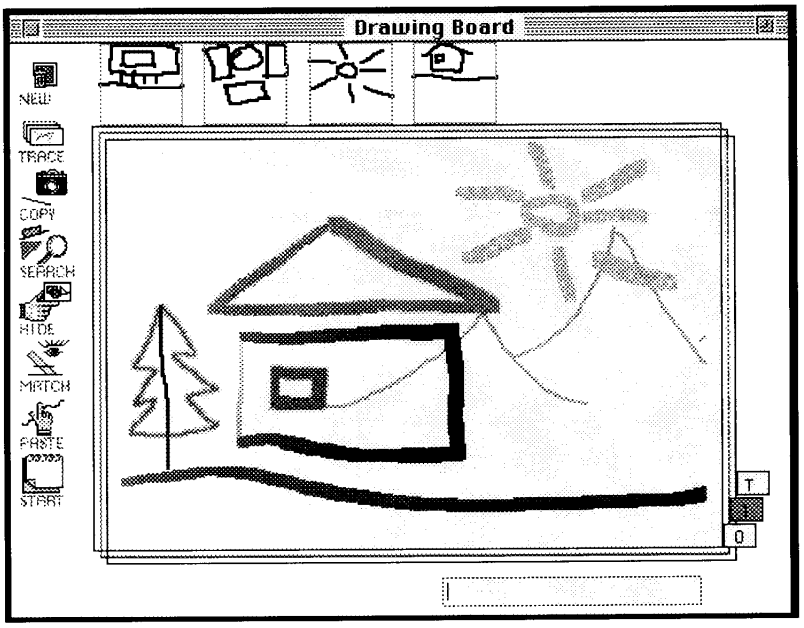

Figure 12: The Electronic Cocktail Napkin

2.7 And So To Informal Interfaces

Mundie and Shultis (1991) consider progress in computer systems. Although many aspects of the concept of "informalism" and "the informal" were presented and examined, there was no appreciation at that venue of its potential application to HCI and user interface design. For example, Reeker (1991), in his paper on *Informalism in Interfaces*, studies some examples of adaptive interfaces, and make an analysis of concepts such as representations of visual knowledge, and projecting cognitive representational structures onto computational representations.

Fisher (1991) examines the question "What is Informalism?" and reaches a number of conclusions. He identifies short-comings with formal methods arising from the nature of the physical versus abstract world, in that they assume finite descriptions, often require completeness, consistency and precision, and are inadequate for describing and reasoning about the physical world. Fisher concludes that, while informal systems must encompass and exploit formal methods, they must be *intensional* and *incomplete*, they must manage *inconsistency*, they must be *nonaxiomatic and prototypical*, and will be *imprecise*.

Meyer and Crumpton (Meyer, 1996 and Meyer & Crumpton, 1996) apply informality to user interface design and architecture. The authors (ibid.) state "researchers and developers are discovering the need and importance of sketch-like representations in the creative process". However, they identify the problem that such applications usually present the user's rough-looking sketches within the formality of a WIMP interface. This creates a problem of a visual jar between the two competing looks: "the 'look' of the computer-generated interface does not reflect the 'feel' of the pen input … (which) is fluid, dynamic, personal and informal, but the computer-generated graphics look linear, static and formal".

Their implementation of an informal interface, EtchaPad (Meyer & Crumpton, 1996), is an interesting one in that the widgets and window shapes take on a rough-looking feel, as if generated by sketch by a human hand. An example of EtchaPad's style of interface is show in Figure 13.

EtchaPad displays two key characteristics of an informal interface as espoused by this thesis, *tolerance* of input and *variability* of output. However, it appears that while an informal interface metaphor is utilised for the front-end graphical display, the back-end (internal)

representations are conventional, and are not in particular concerned with the notion of the cognitive *gist* of the representations.

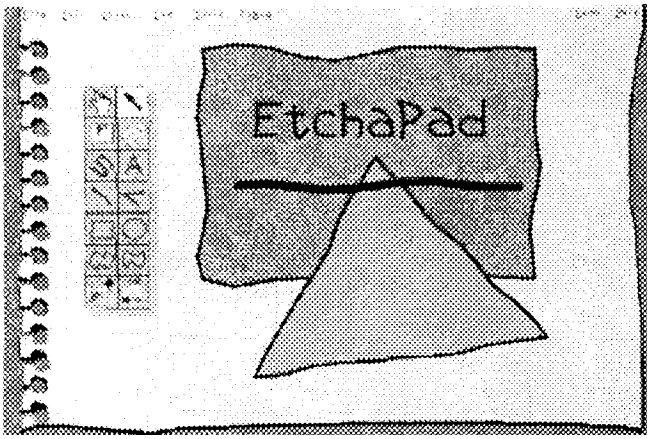

Figure 13: Meyer and Crumpton's EtchaPad (New York University Media Research Lab)

In Chapter 3, the informal interface software that has been developed is described in detail. Any software system that presents its output in a rough sketch-like appearance requires algorithms to generate such sketchy lines, and the algorithms used in this thesis' implementations are described later. These algorithms were developed over a period of time through experimentation, and subsequently tuned to give a more natural and familiar appearance. As a comparison, Meyer and Crumpton (1996) describe a number of methods that they experimented with to try to get pleasing results for drawing informal-looking rough lines, the best of which they decided was to use a *stochastic noise function* algorithm independently developed by Perlin (1985) at the New York University Media Research Lab. Examples of their informal-looking lines and widgets are shown in Figure 14.

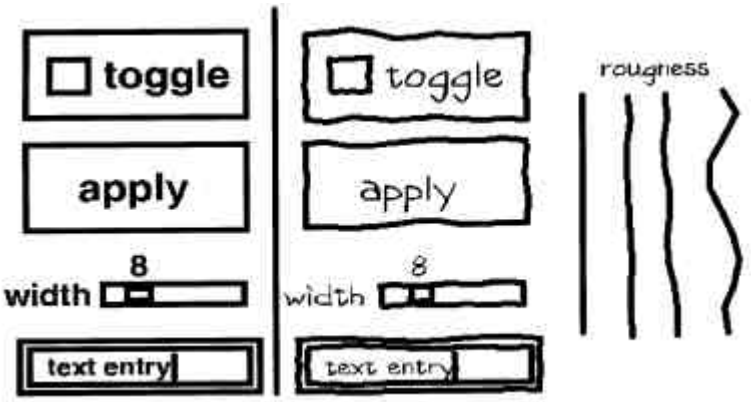

Figure 14: Rectilinear versus Informal graphics, from Meyer and Crumpton (1996)

Meyer and Crumpton's approach is to give a natural, familiar sketch-like appearance to the user interface, and for this the Perlin noise function provides a good solution. This thesis' aim, however, is partly to investigate the underlying cognitive processes inherent in using informality in interfaces, and for this the Protoytpe/Dimension Model (PDM) architecture, detailed later in section 3.6, is more suitable than using other methods such as the Perlin noise function.

Davis et al. (1998) investigate using PDAs such as the 3Com PalmPilot to enable users to take electronic ink-based collaborative notes in free-form way. The authors (ibid.) describe a simple, "informal" (in their own words) system that allows users to take hand-written notes in meetings, and later group them together collectively with other users to create a repository of shared information.

Heiner at al. (1999) have created a hybrid system (the "Paper PDA"), which is a cross between a conventional paper-based organiser, and a fully electronic PDA. This is interesting and relevant to this thesis, because the authors (ibid.) investigate combining the best characteristics of paper and the best of a PDA. They recognise "that paper is a very fluid,

natural and easy to use medium…", yet "has well known limitations when compared to electronic media…". They use a natural, informal, sketch and handwriting interface as input, which later becomes translated into electronic form for further processing.

The subject of *"active reading"* (underling, highlighting and commenting in freehand while reading a text) is explored by Schilit et al. (1998). Their systems uses a large PDA-type tablet with an LCD display, on which a user can display a text and make annotations using a stylus. This is another example of using an informal, intuitive way of interacting with a computer system, and providing input in a sketch-like manner. As well as using this natural, familiar, input mechanism, the underlying computer system further monitors the user's free-form ink annotations, and uses them to discover the reader's interests. So, for instance, the system can search for material related to the annotated text, and display links to this in the margin.

As Long et al. (2000) point out, pen-based user interfaces are becoming ever more popular. One important and desirable feature of such interfaces is the use of gestures (commands issued with a pen) to control the program. The authors (ibid.) examine the design of gesture sets, with the aim of creating a tool to aid in designing sets that are easier to learn. Gestures are interesting in the terms of this thesis in that they are an example of an informal, sketched rendition being used for command and control of a program.

Finally, Igarashi et al. (1999) investigate a potential application for an informal interface as described in this thesis – that of a sketching interface for 3D freeform design. In their paper, they describe the design of a sketching interface for "quickly designing freeform models such as stuffed animals and other rotund objects". The user draws 2D freeform (sketch) strokes interactively, to specify the silhouette of an object. The underlying system automatically

constructs a 3D polygonal surface model based on the strokes. Their program "is designed for the rapid construction of approximate models, not for the careful editing of precise models…". This approach is of interest to this thesis, in that they (ibid.) use an informal, sketch-based interface to allow the user to easily and quickly enter an idea for a design, and then use underlying formal tools and methods (such as polygonal mesh representation) to produce a final model.

2.8 Summary of this Chapter

This chapter has surveyed the literature in a variety of areas – computer science, cognitive science, psychology, ergonomics, computer vision, and art & design. The concept of informality in computer systems, and especially user interface design, is a relatively new one. There has nevertheless been a growing emergence of informality in computer systems and interfaces. The terms "informality" and "informal" necessarily mean different things in different contexts. This thesis is concerned primarily with the application of informality in user interface design and computer systems through the use of tolerance of input, variability of output, and the gist of the underlying representation. These concepts occur in the literature to varying extents, and in various guises, as has been documented in this chapter.

This chapter has further catalogued the evolution of interface systems from the early formal ones such as Unix to the ones described in the latest research papers in sketch-based user interfaces and applications. This chapter notes that the subject of computer vision addresses a rather different problem to that faced by informal interfaces, as described in this thesis. However, it is also noted that much can be gained from the traditional literature of research in areas such as computer vision and graphics, as far as tools and methodologies for constructing

an informal interface as described in this thesis goes. Other well-known constructions, such as *frames* and *objects*, and attributes based on the style of Green's *cognitive dimensions*, and *constraint-based systems* provide a suitable basis for object representations of an informal interface described in this thesis.

In summary, this chapter's survey of the literature finds a trend over time towards various styles of informality in user interface and computer system design. Of particular interest to the thesis is the use of sketch-like input and output in user interface design, and the use of relevant formal methodologies in constructing such an informal interface system. While various elements of an informal interface system as described in this thesis are touched upon in many areas of the literature, this thesis aims to bring several key disparate elements together (i.e. tolerance of input, variability of output, and the gist of the representation) in a novel way.

3. A Framework and Architecture for Informal Interfaces

3.1 An Overview of this Chapter

This chapter sets out a formal framework for representational structures for an informal interface, as described and implemented in this thesis. Although informality is the overriding principle throughout this thesis, as Fisher (1991) points out, using formal methods and putting a formalised structure in place is still desirable. Ultimately an informal interface system (at least, at present) has to run on the formal architecture of a computer and operating system, and so needs to be implemented as an informal layer on top of an underlying formal system.

The proposed architecture described later is meant to be both a workable solution as used in the development of this thesis, as well as a framework for future expansion. The principles at their most primitive level are scaleable to higher levels of complexity, although that is left to future research.

3.2 A General Architecture

This thesis examines the requirements for an informal interface system that, in general, accepts and processes input, transports this in some manner, and then outputs this information again. This could be a locally-based system on a computer, where the input-transmission-output cycle is contained within a single window pane. It could also be a locally-based system communicating between two window panes. Or, it could be communicating between separate computers.

3.2.1 Putting the Framework into Context

As described in this thesis, an informal interface uses some means of sketch for accepting user input, and mimics rough “human-like" sketch for presenting output to the user. For instance, a stylus might be used with a graphics tablet to input data to a desktop computer. The computer might then decompose this input data into its internal representations. This data might then be operated on or manipulated in some way, and then re-presented to the user in a similar matching sketch-like style.

The purpose of the research in this thesis is to examine such an informal interface using sketch to interact with a user, and to further examine the three elements of input, internal representation and output. Thus a framework is required for these three distinct elements. That is, a framework is required for (i) dealing with sketch input from a user and decomposing it into understandable elements, (ii) handling the decomposed elements internally (e.g. storing, indexing or transforming them), and (iii) recomposing and presenting the output to the user.

The implementation described in this thesis draws on the literature of computer vision, but takes a novel approach in practice. This novel approach is taken so that suitable informal cognitive representations are used. Low-level and high-level recognition systems are used, and even higher order levels of systems are also accommodated. The primitive of the low-level system is the Rough Straight Line, i.e. a hand-drawn line. This is internally represented by its primitive (the straight line) and associated attributes (e.g. the shakiness of the hand-drawn line). The high-level recognition system uses an AI approach (using Prolog) to recognise high-level shapes (such as a square or triangle).

3.2.2 Some Types of Informal Interface Systems That Could be Built

This section contains examples of some of the types of informal interface systems that could be constructed.

Later on in the thesis (in Chapter 4) the main implementation is based around the concept of an *informal fax* computer software program. This is a program which works rather like a conventional paper-based fax system, but which uses an implementation of an informal interface. The computer software generates a boilerplate "fax" style form on the screen for the user to fill in with text (via the keyboard) or graphics data (using a stylus). The program would attempt to "recognise" any graphics data input, and if it is able to resolve it to a known informal object (e.g. a sketch of a house, or a sketch of a map giving directions), then it would distil this part of the input information down to the gist of this particular object, for later transmission. (If any data were to be input that the program could not distil down to an informal gist representation, the data can be saved "as is" and transmitted in its original form).

After the user has input all the data, the informal fax program would then transmit the collection of informal gist objects and other "fax" data to a receiving version of the program. This "transmission" could be across a telephone connection as is usual, or perhaps over an Internet connection. This remote version (which need not be exactly the same implementation, in that it could be a simple stand-alone "viewer") would then reconstitute the data in its own style.

Note that as well as the individual object contents being reproduced in the style of the remote program, the overall boilerplate of the "fax" program would be in its own individual style.

However, in order for the program to operate successfully, the viewer must faithfully re-create the original data (in a cognitive, informal interface, sense).

Supposing the user wanted to sketch and fax a diagram of a map of how someone should get to a particular location. There are already many conventional ways of accomplishing this. A person could draw the map on a piece of paper and then physically fax this to someone else, using a fax machine connected to a conventional telephone line. Or someone could sketch the map on a piece of paper, scan it in using a scanner to create a digital bit-map file, and attach this to an electronic mail message to be sent to the recipient. Alternatively, this person could use a conventional computer graphics drawing package to create a graphical image for subsequent transmission by electronic mail.

Using the informal interface "fax" program, the user could sketch in the map using a stylus. The software then decomposes the lines, curves, and other elements that make up a rough sketch of a graph into its internal representation objects. This package of representational objects can then be transmitted to a remote user, and reconstituted by the viewing program.

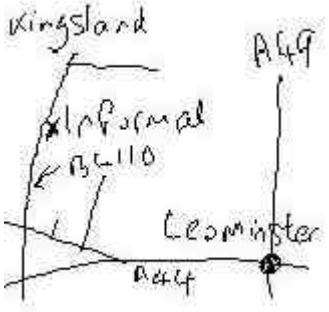

Figure 15:A rough sketch of a map

In this case, what is not necessarily essential is for the reconstituted map to be an exact replica of the original. What is essential, however, is that all the vitally necessary components and associations remain intact. For instance, the directions of the lines (e.g. the roads) must be roughly the same, as must their relationships (e.g. north, south, east or west of each other). Provided all such necessary elements (constituting the "gist" of the map) remain intact, the reconstituted map will provide sufficient information for someone to navigate using it successfully.

Another type of informal interface application might be one to produce informal sketch-like output graphs from data generated by a conventional spreadsheet program such as Microsoft Excel, or a presentation software package such as Microsoft PowerPoint. In this way, what would otherwise be formal renditions of data could be presented more informally, as was discussed previously in section 1.7.1, on page 20.

3.3 Informal Interfaces - A Scope of the Research

This section discusses the boundaries and scope of the research into the notion of applying informality to user interface design and human-computer interaction, and details which areas are investigated. Although much of the research is theoretical in nature (for instance dealing with representational structures), account must still be taken of suitable development platforms and tools for the implementation of demonstration software.

The chapter analyses the potential types of input devices that can be used, such as sound, pen and keyboard, and their suitability for informal interaction. Similarly, output devices are considered. The issue of suitable development platforms, environments and software

development tools is discussed, and finally in section 3.3.2 the actual *systems of focus* - the specific type of informal interaction software programs to be developed - are considered.

3.3.1 Boundaries for the Research

Since there has been little research work conducted into the notion of applying informality to the areas of interface design and HCI, there is much scope in what can be investigated. However, in order to maintain a focus for the research, and to be able to ultimately publish a thesis, there are some boundaries and constraints put upon the exact areas of research.

The focus of this research is at the user interaction level; that is, the direct ways in which a user might input data and needs (operations) to the computer, and the ways in which a computer can output data (results) to the user. Such research is therefore naturally constrained to existing input and output mechanisms (mice, pens, CRT displays and so forth). It might be nice to imagine a different, future, world unconstrained by such "old-fashioned" mechanical input and output devices, but such a world does not exist yet and the thesis addresses informality in *existing* computer systems. The exciting issue of informality in *new* computing is an area left to future research, as discussed in Chapter 5.

Note that the secondary (although equally important) concept of an underlying informal *representation* for such objects and interactions is device independent. As an abstract concept, it is divorced from the mechanics of the actual operations of computing devices.

3.3.2 Systems of Focus

This section analyses the types of areas that are of interest for the research, and that are worthy of closer examination.

There are a number of classes of systems that could be of interest and used as a basis for the study and development of the research into informal interfaces. Examples include music systems (annotation, composition), the architectural design of buildings, or interactive games for children. In such cases, rough sketch-like input can be a natural, familiar and creative part of their functionality.

However, this research is based on the basic premise of (a) sketch input by a user, (b) internal decomposition, recomposition and other machinations by the computer, and (c) simulated "sketch-like" output onto a video screen by the computer. As such, this system is fundamentally of a general purpose nature, but specific examples (for example, the bridge designer application outlined earlier) could be constructed when necessary.

3.3.3 Input Devices

Present day input devices tend to be limited to the ubiquitous mice and keyboards. There has been some move to a greater acceptance of pen input devices, but with limited success until recently - partly because of the poor performance of handwriting recognition software systems. Devices such as the 3Com Palm handheld computer are now proving popular. There are other types of input devices too, such as the 3-D mouse and voice input. Indeed, compared to the rich array of communications devices available to human beings in day to day contact (such as sound, touch, visual cues) the choices in the computer world are still primitive.

3.3.3.1 Development Platforms and Tools

There are, nevertheless, some pen input devices around. But a further problem arises over the suitability of such platforms for the purposes of developing software. Products such as Apple's Newton MessagePad and 3Com's Palm computer are interesting and viable hand-held devices with stylus input, but can be difficult to use as software development platforms.

There are also still a number of PC compatible (that is, computers using Intel x86 microprocessors and running operating systems such as MS-DOS and Microsoft Windows) computers available which are of a stylus input *slate* design, but since they are more targeted at industrial and other vertical markets they are less accessible and more expensive than conventional desktop computers.

3.3.4 Output Devices

Choices of output devices for conventional desktop computers are even more limited than input devices. The only real choice for a graphical output device is a monitor. In a typical IBM PC compatible architecture, the video display memory is addressed through the system bus, with a correspondence between one or more bits of RAM and the colour and position of the corresponding pixel on the screen. Lower level operating systems drivers ultimately directly address the contents of the display memory, setting the bits as appropriate. An alternative display mechanism scheme is used in the X-Windows system, whereby the monitor is a remote client of a server, with its own independent local processing power. Display commands are issued to the monitor from the server over the network protocols.

Another type of graphical output mechanism is a device to draw or print directly onto paper. Examples are a simple printer, a "crab crawler", or a plotter (typically used for the rendition of technical drawings).

Raster graphics displays are driven solely by a pixel bit map. This is obviously a flexible system (within the constraints of the available resolution and size of the overall image), but with a high overhead of required processing power and bandwidth. An alternative display technology to be used is a *vector* one, whereby a command set of vector parameters (e.g. a set of parameters such a "start point", "direction" and "length") is presented to the vector graphics subsystem of the display unit. This is especially suited to the needs of, say, mechanical drawing products, which are typically composed of known geometrical shapes (lines, circles and so forth).

3.3.5 Summary of this section

In summary, this section has examined the choices available for (a) input devices, (b) output devices, and (c) types of operation as a focus for the research, with the following conclusions as to the precise nature of the devices and scope:

3.3.5.1 Choice of input device

Input devices are assumed to be the conventional keyboard, and a mouse or a mouse-equivalent stylus and digitiser tablet.

3.3.5.2 Choice of output device

The output device is assumed to be a conventional computer monitor, preferably a colour one although this is not a requirement. It is also assumed that a graphics display is available, and not just a text (i.e. character) one.

3.3.5.3 Types of Operations

The types of operations that are explored are as follows:

a) sketch input by a user to a computer

b) sketch-like or conventional *crisp graphics* output from the computer

c) some level of interaction between the user and a computer, or between one user and another user via an intermediary computer and/or networking system.

3.4 An Outline Architecture

The architecture of informal interfaces described in this thesis is based on two fundamental concepts: the *primitive* and the *class model*. From these are constructed *informal interface systems.*

Note that the architecture for an informal interface proposed here is particular to the thesis. It is expected that, in general, other informal interface systems would have their own particular architecture. However, the architecture proposed is suitable for informal interface systems using sketch input and output, based on geometrical shapes such as straight lines.

The primitive is the fundamental building block for any informal structure, as addressed in this thesis. Primitives are not necessarily unique in a system, but each system must have at least one primitive.

Class models are constructed from the underlying primitives, depending on the number of primitives available in a system. Every informal interface system requires at least one class model.

For instance, a primitive might be an RSL (a "Rough Straight Line"), and this might be the only primitive in the system. From this a class model can be constructed of all the available geometric shapes that the system is capable of constructing from the primitive, shapes such as a *square*, *triangle* and so forth.

Of course, a class model such as the one above may not be capable of constructing *every* member of the possible members of that class, as this may actually be a superset of members. For instance, a class model constructed of the RSL may be limited in its informal cognitive dimensions, and may only be able to construct members with simple angles (e.g. a rectangle), and not more complex shapes (e.g. a tetrahedron).

Another example of a primitive might be a transistor. The class model could be sets of such primitives connected in various ways, although these might not be very useful electrical circuits without further electronic components such as resistors and capacitors. Figure 16 illustrates a simple class model built on a single primitive.

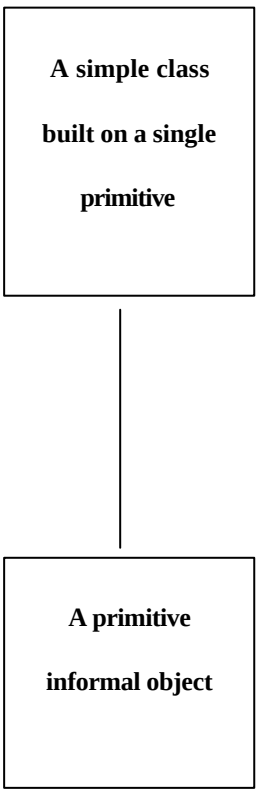

Figure 16: A simple class model based on a single primitive

Higher levels of class models are possible when multiple primitives exist in a system. In this case, more complex objects can be constructed by combining the primitives in various ways. Such a class model is an order of magnitude higher than a class model relying on just a single primitive, as illustrated in Figure 17.

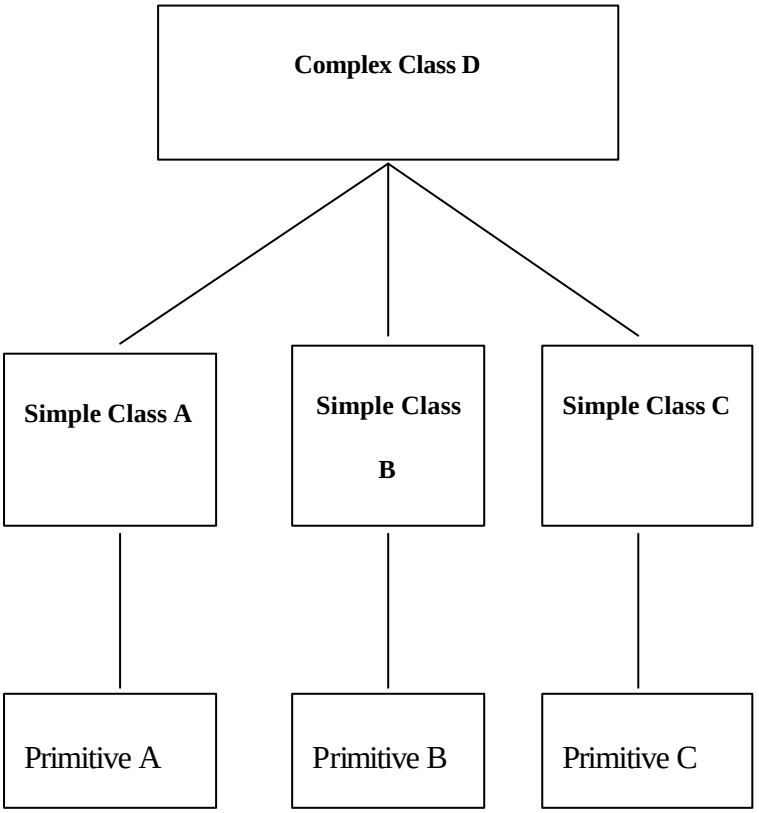

Figure 17: A complex class model based on multiple simple classes

3.4.1 An Architecture for Primitives

The primitive, then, is the fundamental building block. It is based on a singular *prototypical object*, which may be abstract or concrete, to which are attached associated extra *dimensions*, as shown in Figure 18.

For instance, the Rough Straight Line ("RSL") is based upon the prototype object of a *perfect straight line*, which is an abstract geometrical object. Associated with this are a number of *informal cognitive dimensions*, sufficient to fully describe the RSL.

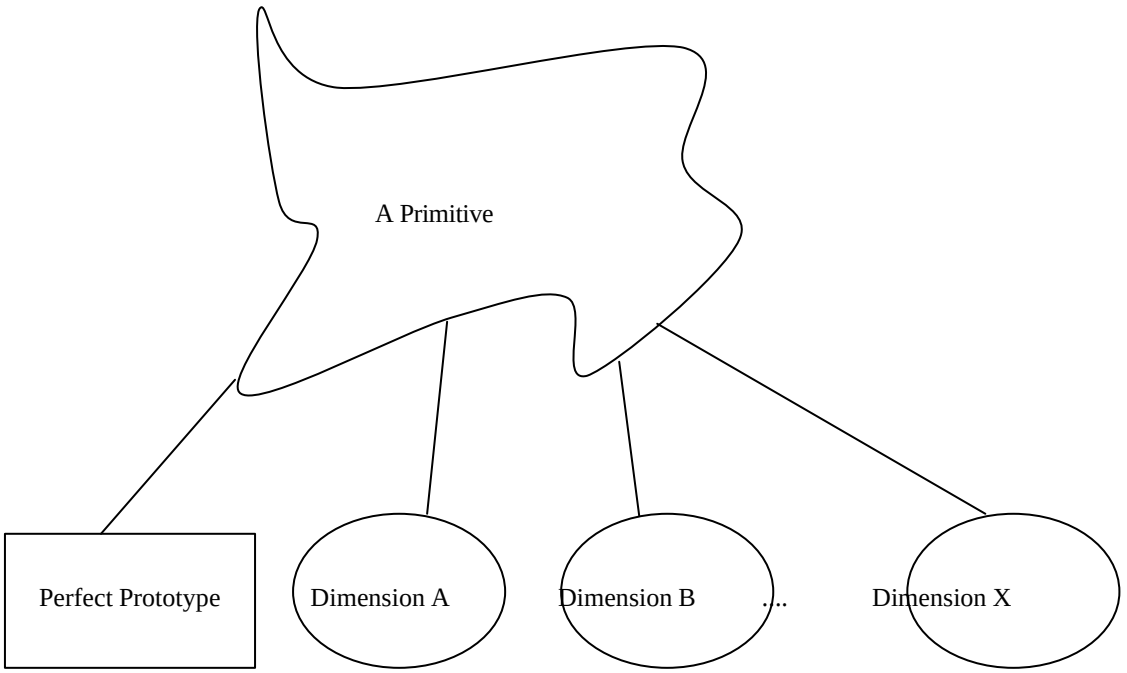

Figure 18: Architecture of a Primitive

3.4.1.1 An Example of a Rough Straight Line Primitive

A Rough Straight Line is composed of the following objects, as shown in Table 1:

Prototype	Dimensions
Perfect straight line	Shakiness Period Direction Thickness Harmony Accuracy Length Start-point

Table 1: The Rough Straight Line

3.4.1.1.1 Definitions of the Associated Informal Cognitive Dimensions

The informal cognitive dimensions associated with a Rough Straight Line are described here; the set was derived empirically through experimentation with software developed for this thesis, as described later in section 4.4.1.

3.4.1.1.1.1 Shakiness

Shakiness is a measure of the pixel variation at right angles from the prototypical straight line, as shown in Figure 19.

Example:

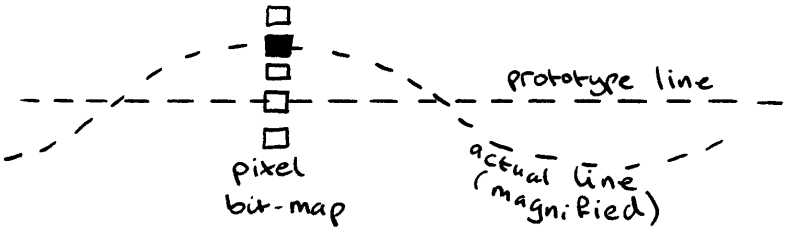

Figure 19: Example of the Informal Dimension "Shakiness"

3.4.1.1.1.2 Period

Period is the average number of pixels, as measured along the length of the prototype straight line, of an average periodic sine wave superimposed on the pixel trace. This is illustrated in Figure 20.

Example:

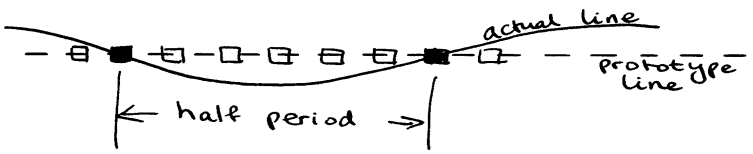

Figure 20: Example of the Informal Dimension "Period"

3.4.1.1.1.3 Direction

Direction is the angle in degrees as measured clockwise from the vertical of the direction of the prototype straight line, as shown in Figure 21.

Example:

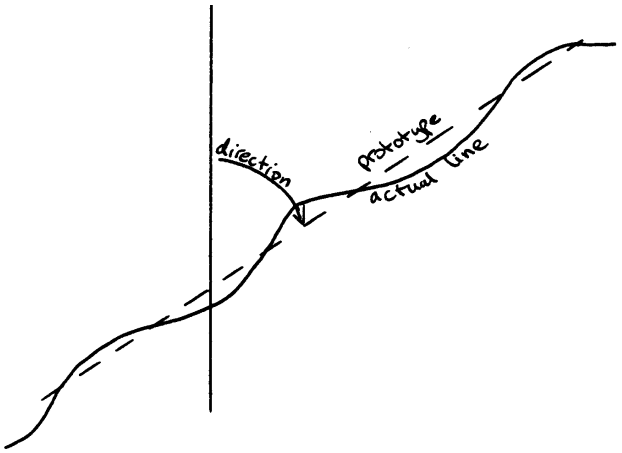

Figure 21:Example of the Informal Dimension "Direction"

3.4.1.1.1.4 Thickness

Thickness is the average pixel width of the line trace, as illustrated by Figure 22. In the implementations described later, this would be the number of pixels drawn in a horizontal or vertical direction by the drawing algorithm.

Example:

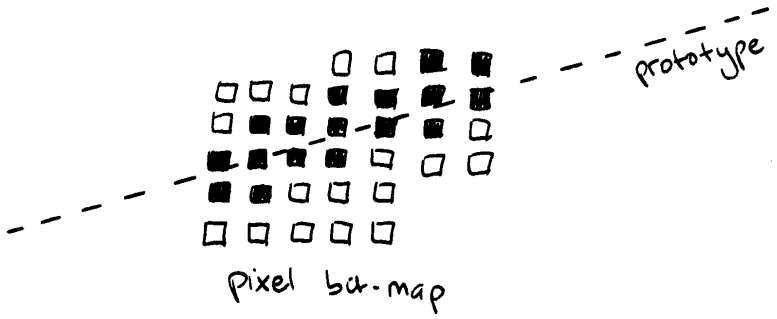

Figure 22: Example of the Informal Dimension "Thickness"

3.4.1.1.1.5 Harmony

Harmony is an abstract concept, based on an arbitrary scale of 1 to 10, of how 'harmoniously' an element fits into a region. For instance, a line drawn in the centre of a sheet of paper is deemed to be harmonious with the region, whereas one pushed over to one side is not harmonious.

Harmony can be deferred and updated. An inharmonious line can become a harmonious square, as more lines are added to orient the final image in the centre of the region.

Figure 23 illustrates the concept of harmony. On the left, a rough sketch of a graph has been drawn quite small (compared to the overall size of the available area) on a whiteboard, squeezed into the top left hand corner. If there is to be no more information added later, and there is no use for the rest of the space on the whiteboard, this is an inharmonious use of the available space. The illustration on the right shows a more harmonious use of the whiteboard.

Harmony may be used instead of *start point*, described later in section 3.4.1.1.1.8.

Example:

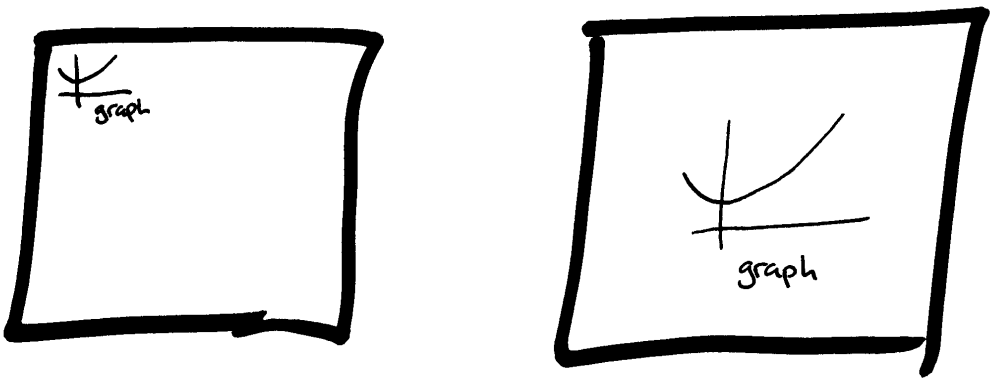

Figure 23: Example of the Informal Dimension "Harmony" - harmonious versus inharmonious sketches of a graph on a whiteboard (context: no further information to be added)

3.4.1.1.1.6 Accuracy

Accuracy is a measure of how well objects are positioned in relation to each other, for instance how well joins are made between two lines. For instance, one way of drawing a square is to draw four separate straight lines, each joined at the end points. An analysis of the average pixel discrepancy in the accuracy of starting a new line near to or exactly at the end of another line is a measure of the accuracy. This is illustrated by Figure 24.

Example:

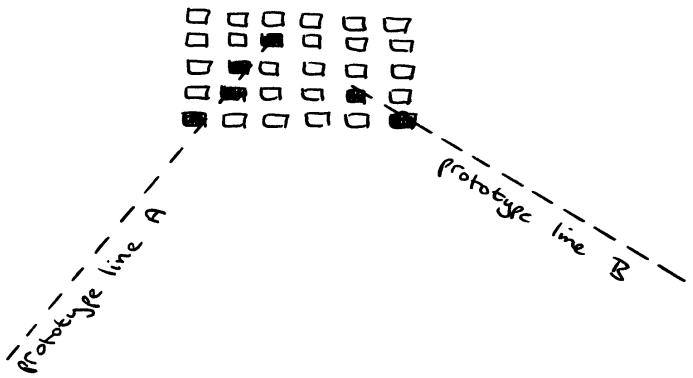

Figure 24: Example of the Informal Dimension "Accuracy" - line A 'joined' inaccurately to line B

3.4.1.1.1.7 Length

The *length* is the number of pixels that would be traced out by the prototype straight line, as shown in Figure 25.

Example:

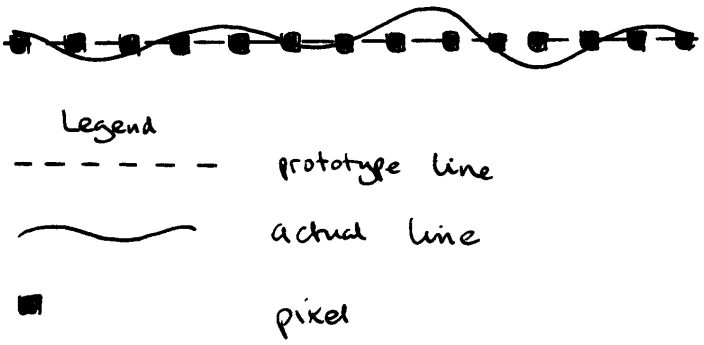

Figure 25: Example of the Informal Dimension "Length"

3.4.1.1.1.8 Start Point

The *start point* is the pixel location, as measured from an invariant location in the region, of the start of the (virtual) trace of the prototype straight line, as illustrated by Figure 26. This pixel co-ordinate is a prototypical one, as it may become physically altered by *accuracy*.

Example:

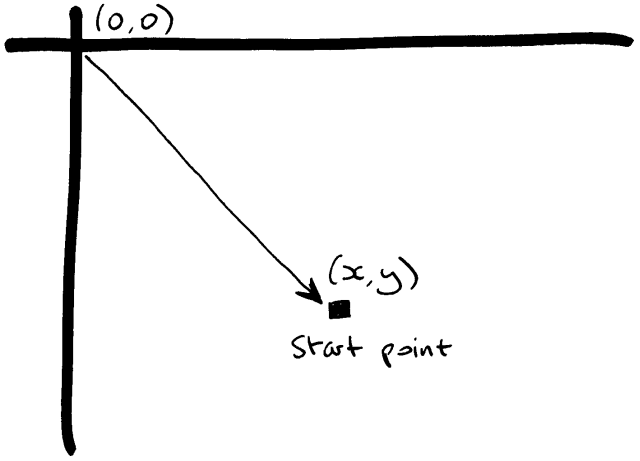

Figure 26: Example of the Informal Dimension "Start point"

3.4.2 Functional Levels of Complexity - Class Models

In order to differentiate between informal interfaces embodying differing levels of functional complexity, a system of Classes is used based on the core set of fundamental informal object primitives available (conceptually or physically) to the recognition and inference engines. Note that different functional engines in a single implementation could be based at different class levels.

Class Model levels are assigned according to the complexity of the fundamental primitives. A *Class 1* model utilises a single primitive object (e.g. the *straight line*). This is the fundamental and only primitive that is understood; that is, a decomposition engine recognises only the Rough Straight Line as a valid informal object. All other objects are maintained in their original form, e.g. as a pixel vector trace.

An architecture operating at a class higher than *1* has the capability of resolving objects to a higher level of abstraction. For instance, in the implementation described here the recognition engine runs as a *Class 2* architecture: extra higher-level abstractions of primitives such as *triangle* and *square* are recognised. A Class 2 architecture is one which runs above Class 1 and which resolves higher-level abstractions that are composed of a *single* Class 1 object. A Class 2 architecture needs to rely on only a single Class 1 architecture as it builds all its (more complex) primitives, such as squares and triangles, from a single Class 1 primitive - e.g. the straight line. A Class 2 "primitive" becomes a basic primitive in its own right, although it is composed of primitives from lower classes.

A *Class 3* architecture resolves Class 2 objects (and hence implicitly Class 1 objects too), and which also has the capability of resolving to an even higher level of abstraction. A Class 3 implementation might be able, for instance, to recognise complex shapes such as *house*, *dog* or *train*.

Note that a Class 3 model is actually composed of *multiple Class 2 models*, and hence implicitly multiple Class 1 models too. Thus a suitably complex recognition model at Class 3 can always be built from a sufficient number of Class 2 models; by inference, any unresolved object can *always* be adequately resolved by the addition, at worst, of a new Class 1 model.

There is an important difference between these three classes of architecture, both conceptually and in implementation. For instance, it is possible to build a software application (and such applications are described later) that seemingly operates at a Class 2 level, in that it is operating seemingly easily at a level of squares and triangles. However, analysis of the underlying engine would show that only a Class 1 architecture (e.g. just a straight line) is being used. A Class 1 architecture can itself be quite powerful. However, the same results (seemingly) can be accomplished by a Class 2 architecture, but again closer analysis would show that a higher level of abstraction (e.g. squares and triangles) was genuinely being utilised internally. That is, a Class 2 architecture accomplishes the same end result, but at a higher level of abstraction.

As an example, consider a Class 1 implementation in which the Rough Straight Line is the primitive. A square in such a system would be four such lines, i.e. *four* instantiations of the primitive. A Class 2 implementation using a Rough Square as its primitive would actually represent this square as a *single* instantiation of the primitive. The two present the same representation at a high level, although they use different underlying levels of representation.

3.4.3 A Problem for Class 2 Models

There is one common problem that Class 2 models face: resolving conflicting abstractions. For instance, is the drawing as depicted in Figure 27 meant to be two squares joined together, or a rectangle with a bisecting line?

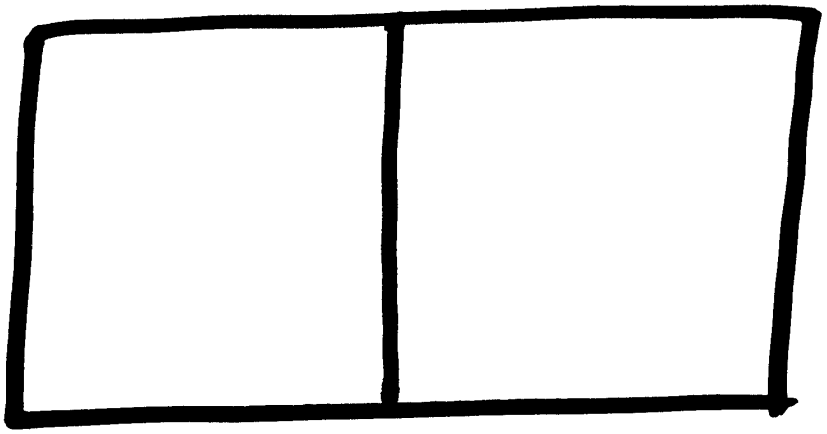

Figure 27: A Resolving Problem for Class 2 Models

This problem differs from that of Computer Vision, however, in that the input stream from the user is temporally dependent (i.e. it reflects the order in which lines are input). Thus the recognition engine may have valuable clues as to what the user might have *meant*, although this may be difficult to always determine. For instance, in the example above the input engine would have logged the order in which the lines were drawn as follows, as illustrated by Figure 28:

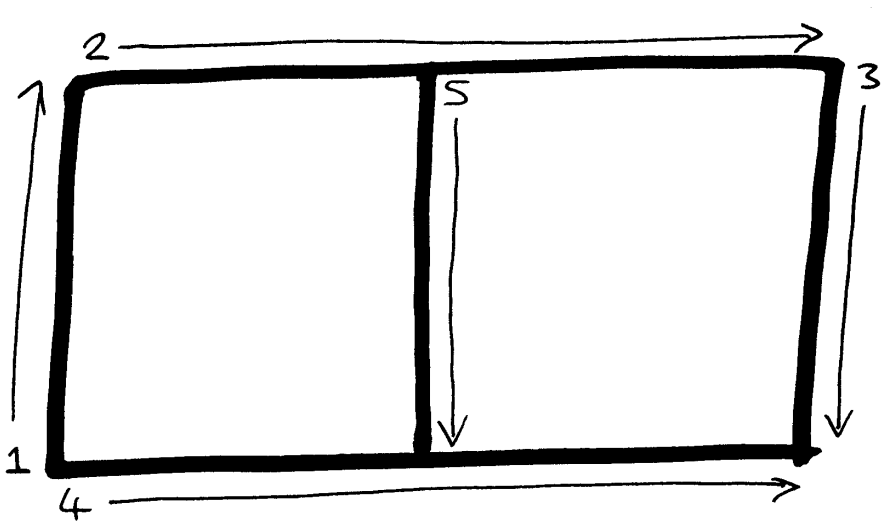

Figure 28: Trying to Resolve the Squares Problem

However, this does still not mean that the user meant the sketch to be that of a bisected rectangle; it could still be two squares joined together, even though its 'bisecting' line might be expected to be the result of different strokes (e.g. 1, followed by half of 2, followed by 5, and back to the start of 1).

This is of relevance to this thesis because of the way that the higher-level recognition engine works, as described in this thesis. The implementation described would first resolve the line segments one by one, as the user drew them in, into Rough Straight Lines. This task is performed by the low-level object recogniser. These line segment objects are then passed to the high-level recogniser. So the example in Figure 28 would be resolved as a rectangle 1-2-3-4, bisected by line 5.

Alternatively, a high-level recogniser that is more sophisticated than the one described in this thesis could be implemented, which would take into account the fact that such a drawing as in Figure 28 could actually be two squares, sharing a common side.

3.5 Fundamental Primitives

Any Class 1 model architecture engine recognises a single type of primitive object, and hence a single prototype object. This object need not necessarily be simplistic however. In the described implementation the fundamental object is a *Rough Straight Line*, whose prototype is not the most primitive potential object in that a *point* is more primitive. Indeed, the Class Model architecture accommodates this, as *any* pixel bit map object can be resolved by using a Class 1 model which has the point as its primitive.

Thus one group of Class 1 models could recognise a single Euclidean geometric shape, such as a *line* or *circle*. A Class 1 model could similarly be based upon a more complex geometrical shape, such as a *parabola*.

Note that different taxonomies or groups of Class 1 models exist, depending on the base architectural principle (Euclidean geometry, for instance). Class models could be constructed using, say, fundamental electronic components, or architectural (in the sense of the design or layout of buildings and rooms) objects.

3.6 The Prototype/Dimension Model

The construction of the fundamental primitives described above can be succinctly termed a Prototype/Dimension Model (PDM), in that an object is composed of a single primitive prototype, attached to which are one or more informal cognitive dimensions.

3.6.1 Operators on PDMs

Although a PDM object is actually composed of a number of elements, it can be considered as a single unified object for some purposes, such as the applying of operators. For instance, geometrical operations can be performed on a geometric PDM, such as *rotation* or *reflection*. Constraints such as *attachment* can be applied to pairs of PDMs, and then in turn operators can be applied to sets of PDMs. In a constraint-based implementation, an operator would be applied to a single PDM, and *propagation* ensures that its effect is applied to all other associated (i.e. constrained) PDMs. So, for instance, if we wish to rotate a rough square composed of four rough straight lines, if those lines are constrained in the representational

model (through *attachment* at their end points by specific angles), then we need to apply only a rotation to one rough straight line for the whole object to be rotated.

Constraints are represented in the architecture by further informal cognitive dimensions. For instance, one constraint linking two adjoining lines in the square would be *'attached to'*. A further constraint linking the adjoining lines would be *'at right angles to'*.

3.7 Representational Conformity

One notion concerning the presentation of the output in this thesis' implementation of an informal interface is that of *Representational Conformity*. Although allowing the program which is generating the output images much flexibility in the way it can choose to interpret these informal objects is a keystone of the architecture, there are bounds within which this variability must be constrained.

A similar issue is currently being faced by the HTML description language. Although originally the multitude of Web browsers were able to freely interpret, lay out and display images, the move in recent times (with the emergence of just two competing browsers on the market and the inherent commercial demands) has been to tighter constraint in layout and positioning possibilities. There has also been understandable demand from Web page designers to have more control on how their designs will appear ultimately on the screen.

3.7.1 Representational Conformity and Informal Interfaces

The software developed for this thesis was under the control and guidance of a single developer, but this would not be the case were such software available commercially, when

there might be many different software developers with their own interpretations of image reconstitution.

In the above case study of an informal fax viewer, there would be a number of different levels of interpretation to be considered:

a) reconstitution of boiler plate *forms* (i.e. the layout of the 'fax' page)

b) re-layout of text items (e.g. fonts, sizes etc if not specified)

c) reconstitution of informal objects

In case (c) the developer implementing the software would have wide scope in interpreting images such as *house, box, bridge* and so forth, depending on the highest level of object abstraction being utilised. This can be used to good advantage in locale dependence (an American house being made to look different to an English one, for instance), and is a natural by-product of gist interpretation.

3.7.2 Style Guides for Representational Conformity

We would expect that in some cases of particular implementations guidelines would be published (akin to the *Style Guidelines* for user interfaces published by companies such as Apple Computer Corporation) to ensure that there would be some level of conformity of interpretation of Class models across platforms or different implementations. With the nature of gist being dependent on context and culture, then these style guidelines may also need to be context or culture dependent. For instance, renditions of rough sketches of houses may be dependent on the country in which they are based, and English houses tend to look different

from American ones. So in order to retain the gist it may be necessary to be aware of the context in which it is currently based, although this may not always be the case.

3.8 Summary of this Chapter

This chapter has set out a general architecture for a type of informal interface addressed by this thesis, i.e. one that allows for accepting sketch input from a user, performing an analysis on that input data so as to decompose it into primitive elements, then performing a higher-level analysis of that data, and finally recomposing and re-presenting the data to the same or another user. The implementation in this thesis uses the construction of a low-level recogniser and a separate high-level one. The low-level recogniser works at the level of the *Rough Straight Line* (*"RSL"*). The high-level recogniser analyses these RSL primitive objects, to try to recognise higher-level constructions such as squares and rectangles.

This chapter also considers some types on informal interface systems that could be built using this architecture, such as a sketch-based "informal fax" system, and an "informal graph" generating system. A scope for the research in this thesis is then considered, in as far as the types of input and output devices that can be handled. Various types of input and out devices are discussed, and their suitability or otherwise considered as far as the requirements of the thesis go. Ultimately, the implementation described in this thesis uses a conventional mouse, keyboard, and digitising tablet as input devices, and a colour graphics display monitor as an output device.

This chapter then proposes an architecture for an informal interface as described and implemented in this thesis. This is based on the principles of *the primitive* and *the class*

model. Although other primitives could be considered, the only one handled in the implementation in this thesis is the *Rough Straight Line*. Class models are constructed from the underlying primitives that they contain. A primitive is composed of its singular *prototypical model*, and its associated *informal cognitive dimensions*. The combination of primitives and associated informal cognitive dimensions, and class models, is termed the *Prototype/Dimension Model ("PDM")* according to this thesis.

Finally, the issue of *Representational Conformity* is discussed, that is, "style guidelines" for informal interfaces as described in this thesis. This chapter proposes that in order for such a type of informal interface to be useful, due regard needs to be given to how class models and primitives are handled and rendered across different implementations of such systems.

4. The Development of Software Tools and Applications

4.1 An Overview of this Chapter

This chapter describes some example software applications to investigate and demonstrate informal interface concepts.

A number of software components need to be constructed for an informal interface software system. At the outset, the capabilities of the input and output mechanisms must be investigated, as well as how to effectively represent and handle the informal images. So, as far as the input mechanism is concerned, it must be able to accept the graphical input of a drawing device such as a mouse or a pen in some way, render this on the display device, and store the input in a useful internal representation.

In the case of input, suitable software is constructed to accept the point co-ordinates as data from a device such as a mouse.

As far as the output mechanism is concerned, two elements must be considered: (1) the actual rendering of the informal image, and (2) a mechanism to create this image from the internal representation. So, it must be able to render a suitable output image (as a pixel bit map on a conventional computer graphics display) from an internal informal representation. Since the output image needs to look informal, in the sense that straight lines look like hand-drawn *rough* straight lines, suitable algorithms for creating effective looking output images must be devised.

Having established these essential building blocks, more elaborate informal interface systems can now be constructed. In this case, the components are split into a number of different areas:

1) input of the (graphical) image

2) a low level decomposition engine to resolve the input image into primitive informal objects (e.g. rough straight lines plus informal cognitive dimensions)

3) a higher-level analysis engine to attempt to resolve low level primitive objects into higher level constructs (e.g. a number of rough straight lines may make up a higher level primitive construct such as a *square*)

4) a method for transporting high or low level informal objects from one device or user to another device or user, across any arbitrary transport mechanism (e.g. file copy, TCP/IP, or modem link)

5) an independent method for reconstructing a rendition of an informal output image from the low level transported primitives

and finally

6) an independent method for reconstructing a rendition of an informal output image from the high level transported primitives.

Note that (5) and (6) are entirely different cases. In the case of (5), simple low level primitives (i.e. rough straight lines) are merely reproduced; providing there is no error in decomposition or transportation, the output image will be *CI-equivalent*. In the case of (6), a high level

informal object (e.g. a “square”) is rendered according to the “understanding” by the output engine of what a square should be.

These software programs have been developed in a variety of languages (such as ‘C’, C++, and Prolog) depending on the design requirements, under a number of development systems - principally Borland C++ Version 4.5 (Borland, 1992) and Cogent Prolog (Amzi, 1995).

4.2 Choices for Representational Structures

In constructing an architecture for informal interface representational structures there were a number of different ways of accomplishing the task, which are examined in more detail here.

4.2.1 Frames

Using *frames* (Minsky, 1975) as a representational framework is appealing; an informal object lends itself well to being represented by a frame and fillers - the skeletal frame being the fundamental primitive such as a straight line, and the fillers being the associated informal cognitive dimensions such as shakiness, thickness, colour and so forth.

However, development tools and environments are not so prevalent for creating software using frames, one notable exception being Apple Computer’s development environment for the Newton MessagePad.

4.2.2 Objects

The representations are, naturally, object-based, although the term *object* can be interpreted in a number of ways. In a strictly programming sense, such as in C++, an object has a precise

defined meaning. As such informal objects could certainly be couched in terms of a C++ object; although if the representations of the informal objects themselves are not couched in object-oriented terms then the extraneous development can certainly be so. That is, in the PDM architecture, an informal interface object (such as a rough straight line) is an "object" in the sense that it is an encapsulation of data. So these "objects" could be physically programmed as C++ "objects", or in a proprietary data encapsulation manner in another programming language.

4.2.3 Prototype/Dimension Models (PDMs)

In fact, a particular implementation of PDMs was used, being a hybrid frame/object construction, but based on a (deliberately) verbose ASCII character architecture.

4.2.3.1 PDMs versus Objects

In a refined implementation of software for this informal interface system, it might be preferable to use a more compact object-oriented structure for the internal representations than the ones used in this thesis. However, in the implementation described, a verbose representation is utilised. In an earlier implementation this took its style from HyperText Markup Language (HTML) used widely in the Internet, based on the use of ASCII text based files. An advantage is that ASCII text files are easier to browse, study and understand, without the development of separate browser programs. And, as with HTML, they are extremely portable across networks and different platforms.

For convenience, in the current implementation Prolog predicates are used to define the PDM elements. This choice was made so that the output from one component could be fed directly into the Prolog-based component, without further translation.

4.3 A Set of Informal Interface Tools and Applications

One tool is a Microsoft Windows based utility called the Informal Interface Object Browser (I2OB), developed to investigate the generation of primitive informal objects (i.e. primarily the fundamental rough straight line or *"RSL"*) and associated informal cognitive dimensions. This program provides insight into a possible set of dimensions on which to base informal objects. The program is detailed later.

A more sophisticated application takes the form of an *informal fax* or *email* type program (called "**i-Fax**"**)**, in which the user interacts with the program to input data to be "faxed" in a number of distinct ways: (a) conventional keyboard typed input, (b) conventional graphics input (e.g. handwriting script), and (c) other ("understandable") graphics input interpreted and stored as *informal objects.*

There are a number of other support programs to illustrate the use of informal objects: (1) a transport layer handler, (2) an independent alternative object viewer, (3) an analyser engine and (4) a post-analysis reconstructor. A structure for this architecture is shown in Figure 29. In the implementation described in this thesis, the i-Fax program is represented by the "informal fax application i-fax" and "Decomposition Engine" boxes. The Xport program is represented by the "Transport Layer" box. The DOSView program is represented by the "Simple Object Viewer" box. The Prolog program EXAMINER is represented by the "Analysis and

Recomposition Engine" box. The program i-View is represented by the "More Complex Object Viewer" box. These components are detailed later, but by way of an outline illustration a session would typically consist of a user first constructing a "fax" using the i-Fax program, which would be a mixture of text characters, and graphics images. When graphics images are input the program attempts to interpret the graphics vector trace into a primitive informal object (i.e. a "straight line"). If a successful decomposition is achieved, the analysed object is stored in a temporary output file, as are text characters and miscellaneous (uninterpretable) graphics traces.

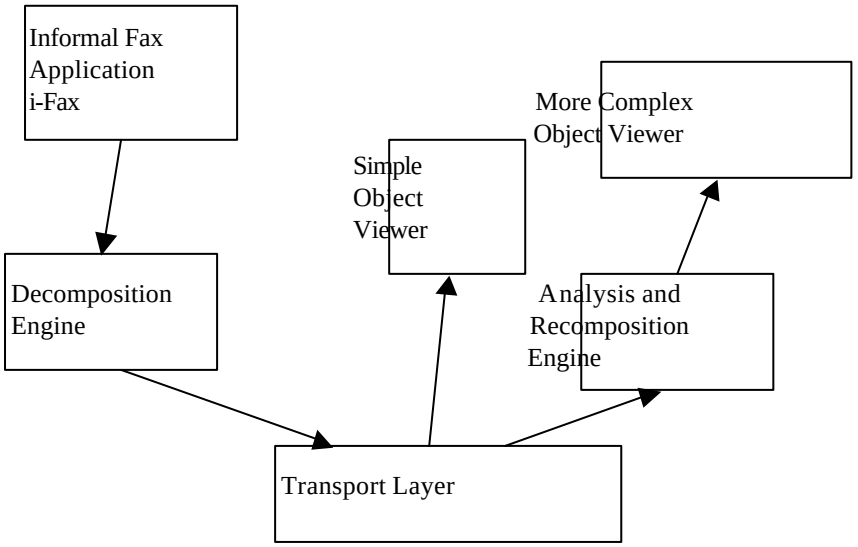

Figure 29: Block diagram of an informal faxing system

When the user has finished composing the "fax", the **send** function causes the program to concatenate object descriptors into a single output file, which is, for instance, spooled to a **\OUT** subdirectory.

A second utility, the **transporter**, can then be used to convey the message file. In this illustrative case the transport merely copies the file from one subdirectory location to another;

in principle the file could be embedded with an email, compressed to a binary stream, sent over a modem link, converted to HTML extensions before transmission, or whatever.

An example of an independent informal object viewer, **DOSView** (it is so-called "independent" since it is an entirely different development from i-View of the interpretation and reconstruction of informal objects, and indeed was developed using a different programming language and development environment to i-View) can at this point be used to view the resultant "fax" output.

EXAMINER is another more sophisticated informal object interpreting program, applying AI programming techniques to analyse the set of informal objects. This program was developed in Prolog. Examiner can, for instance, recognise and extract an (embedded) interpretation of simple graphical objects such as *a square*. This interpretation is re-spooled out as a post-interpretation representation file, using an HTML-style or Prolog syntax.

Finally, a viewer **i-View** can be used, perhaps by a distant and different user who has received this informal fax, to view an interpretation of the object sets. In this example case, the user might have entered some text and a drawing of, say, the rough design of a house. i-View attempts to reconstruct the small informal object primitives that it finds in the input file, into (hopefully) the same looking house that was originally input - at least maintaining the gist of the representation to all intents and purposes. Its metric of success (a concept which is examined in more detail later) is the degree to which the original gist of meaning is retained from the original to the reconstituted interpretation.

4.4 A Detailed Description of the Software Tool Set

The software tools described in outline above are now examined in more detail.

4.4.1 The Informal Interface Object Browser, I2OB

I2OB, an informal object constructor, is an initial investigative tool developed for informal interfaces. It is built as a Microsoft Windows 3.x application and written in C. This program is able to build some informal interface objects out of the single primitive of a *rough straight line*; from this foundation more complex objects can be built, such as rough graph axes with functional relationships (i.e. a straight line or other more complex curve graph depicted a relationship such as $x=2y$ or $y=x^2$), squares, triangles and grids. The basic *prototype* is the perfect straight line, and its form is varied by changing the *informal dimensions* of this object or frame. The informal dimensions handled are *shakiness*, *period*, *direction*, *thickness*, *harmony* and *accuracy*.

The purpose behind this program is to be able to experiment with different informal dimensions, to discover which ones were necessary, interesting, and possibly interrelated, and to develop algorithms for their construction. For instance, the program was initially constructed with only the ability to draw a perfect straight line in a fixed direction. This was then extended by further programming with the concept of *shakiness*. Shakiness is indicated by a number on a scale of 1 to 10 that was factored in to a random number generator to create variability in the displacement of the next pixel to be drawn from the prototypical straight line. This was not found to produce convincing, "human sketch-like" results, so the dimension of *period* was introduced (a carrier sine wave), again factored in on a numeric scale. This created more

convincing results, in that the resultant lines looked more like ones drawn by humans using a mouse or stylus as input.

Similarly, other dimensions were introduced and discarded or incorporated after experimentation. Thickness is the average line width in pixels. Direction is the angular displacement from the vertical of the prototype. Harmony is a measure of how 'harmoniously' the object fits in with its environment, i.e. how well, say, a triangle is placed in the screen to balance available space and size. Accuracy is a measure of how well the start of one line, for instance, is joined up with the end of a previous one. Figure 30 shows an example of a rough sketch-like square drawn by I2OB. The dimensions for the last object drawn are shown at the top of the screen.

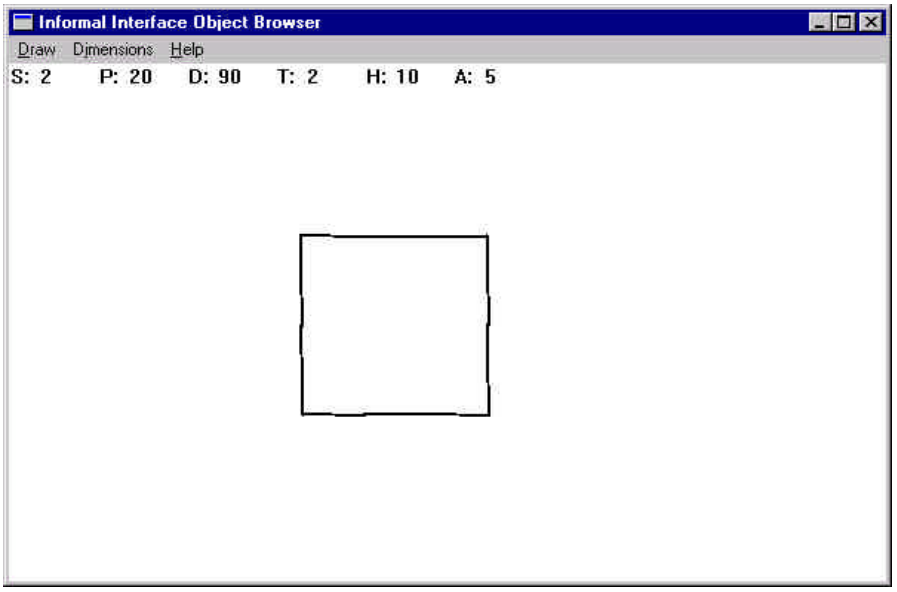

Figure 30: An example of a square as drawn by the Informal Interface Object Browser

Figure 31 illustrates how shapes can be chosen for drawing by I2OB.

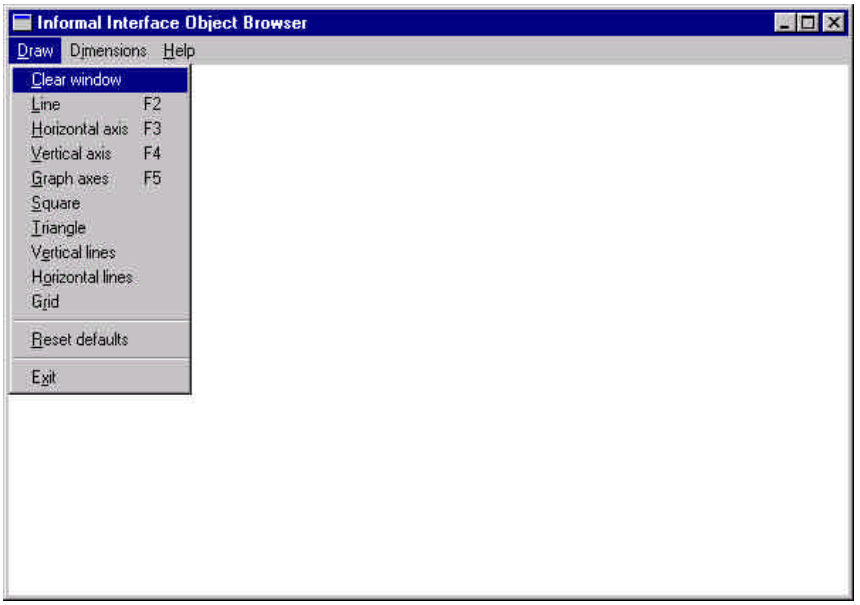

Figure 31: Choosing a shape to draw

Figure 32 shows how the informal cognitive dimensions can be changed.

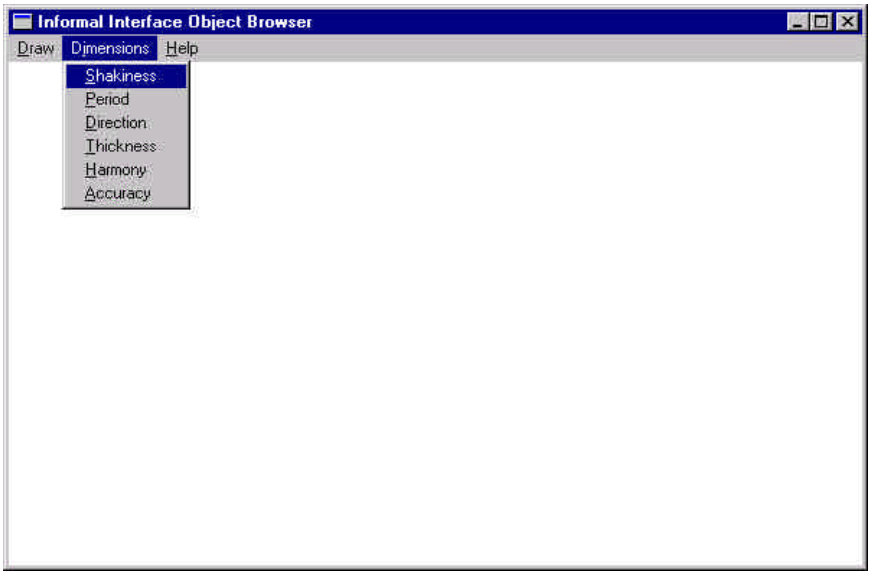

Figure 32: Changing the informal cognitive dimensions

Figure 33 illustrates a square drawn with slightly different dimensions: the level of shakiness is now 3.

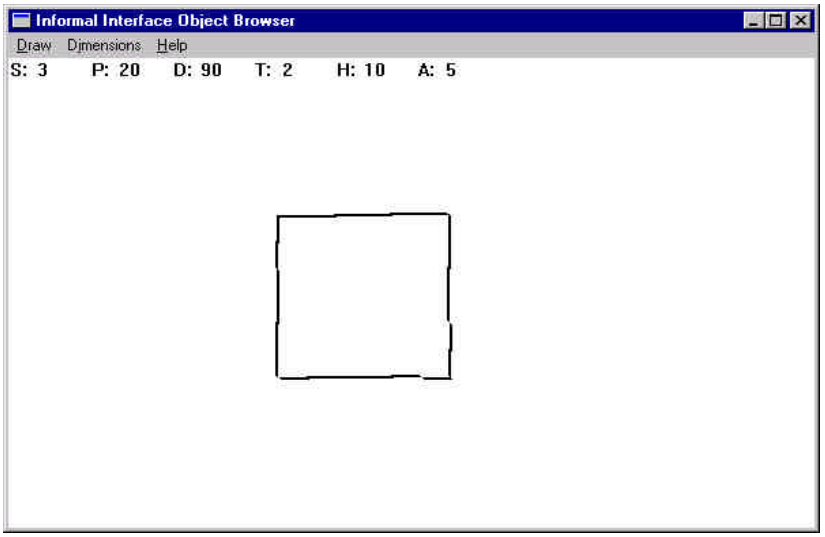

Figure 33: The effect of increasing shakiness

Figure 34 shows the effect of increasing the thickness of the line.

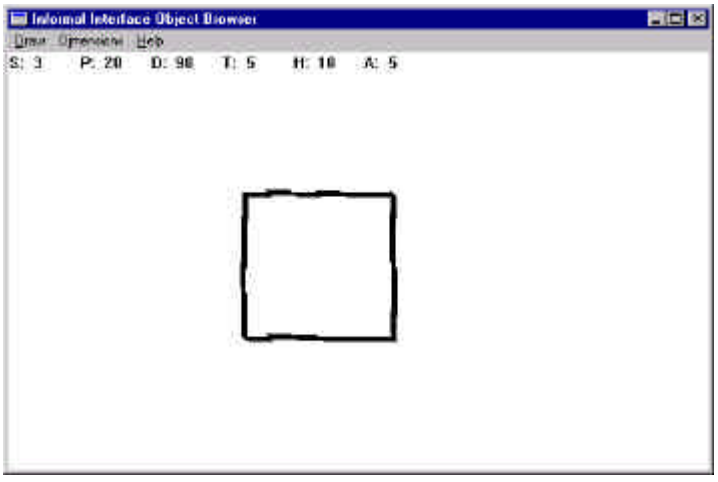

Figure 34: Increasing the thickness of the line

Figure 35 shows another example of a square. The informal cognitive dimensions are the same as in Figure 30, but the pixel bit-map trace of the lines of the box is slightly different. In a particular context these two sketches of a box are CI-equivalent.

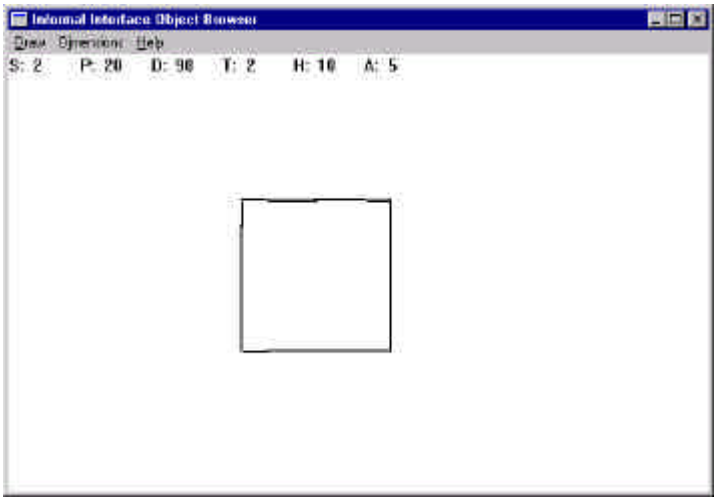

Figure 35: Another example of a square drawn using the same informal dimensions

With this simple constructor, we can build straight lines or more complex objects and compare them with how a human might be able to draw them given a suitable stylus and touch screen. For instance, a combination of informal dimensions can be found to give convincing images, but changing just one of the dimensions a little may make it immediately obvious that it was not drawn by a human (a “straight” line might start to look too shaky, for instance). However, we have found that when drawing a more complex object such as a grid composed of horizontal and vertical lines no amount of modifying the current set of informal dimensions can create a convincing picture; we need to add more dimensions (such as δ direction or change in the basic direction, or variability from a more complex primitive such as a perfect grid as opposed to variability from its own *atomic primitives*, the straight lines). Although the gist of the representation is retained (i.e. a “rough grid"), generally when users were asked if the image

had been created by a computer or a human they guessed that it was computer-generated. When asked why, it was because although each individual line was convincing and like a human-drawn sketchy one, the overall effect was too ordered and precise. There was not quite enough variability in the output - i.e. not enough variability in the spacing of the prototype grid lines.

4.4.2 The Informal Fax, i-Fax

i-Fax mimics a software fax style program in which the user is presented with a pro forma fax page layout. Ultimately, a user would be able to input data in three ways:

1) text (character) input, using a keyboard

2) handwriting cursive script, to be analysed by a handwriting recognition engine

3) sketch input, to be analysed by an informal interface engine.

Any input not recognised by the above three methods would be saved as a pixel bit-map or vector trace.

At present, the implementation of i-Fax handles only case (3), which is the focus of the thesis.

The implementation of the program attempts to recognise one type of primitive: the rough straight line. If it recognises input as being a rough straight line, it resolves the line into its PDM of the prototype straight line and associated informal cognitive dimensions. This is does by means of a simple best-line-fit algorithm, as detailed in the Appendix A software listings, with the addition of post-processing to extract the supplementary informal cognitive dimension objects or *fillers*. A successful decomposition and resolution process would thus result in a

vector trace input being resolved into an informal object text analysis output file, spooled individually into an output subdirectory for later concatenation and reprocessing. The following is an example of an output descriptor for a particular rough straight line:

```
%Informal-object-structure-version: 1.01
objecttype(hAMAEAGFO, line).
shakiness(hAMAEAGFO,6).
period(hAMAEAGFO,2).
direction(hAMAEAGFO,90).
thickness(hAMAEAGFO,1).
harmony(hAMAEAGFO,0).
accuracy(hAMAEAGFO,5).
length(hAMAEAGFO,203).
startpoint(hAMAEAGFO,267,372).
```

Notice that the syntax is:

<informal cognitive dimension>(<unique handle><value>(<value>))

The i-Fax implementation is also currently used as a vehicle for some extra experimentation and debug purposes. For instance, it can generate its own rough sketch-like lines, and more complex images such as a *square* and a *house*. An example of a sketch-like house generated by i-Fax is illustrated in Figure 36.

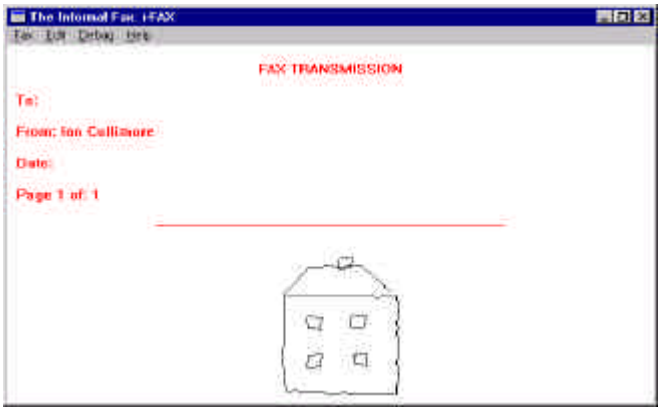

Figure 36: A rough sketch-like house as generated by i-Fax

Notice how changing the informal cognitive dimensions will alter the image. In Figure 37, shakiness has been increased to such an extent that the drawing is no longer convincing.

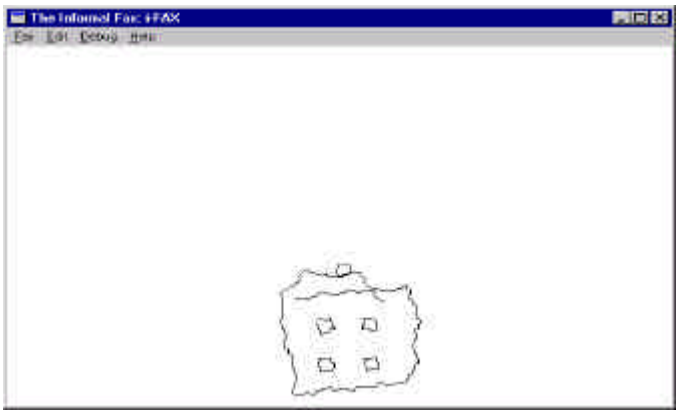

Figure 37: The effect of increasing *shakiness* in i-Fax

Also, i-Fax can be used to dump information concerning objects that have been entered.

Figure 38 and Figure 39 show dumps of the data for particular hand-drawn lines.

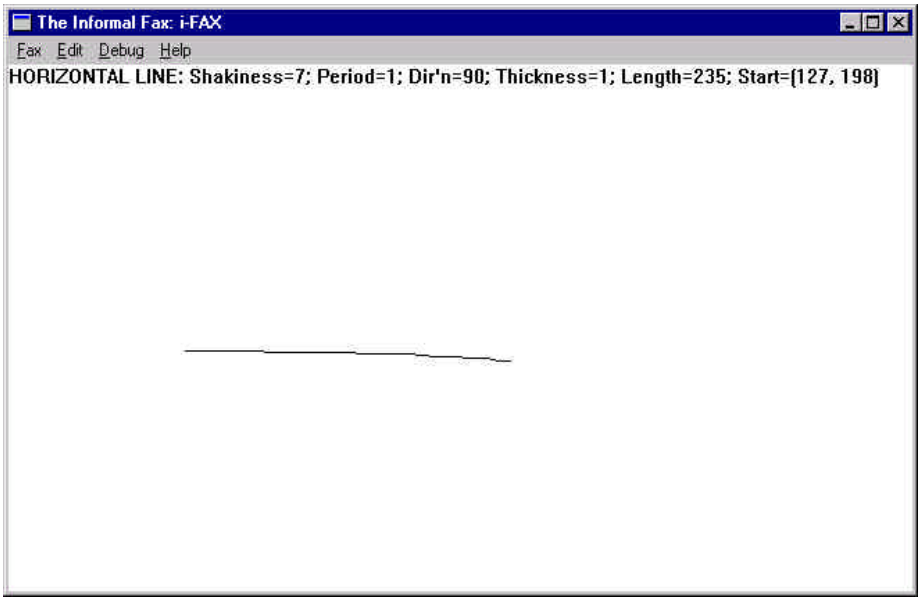

Figure 38: A Debug Data Dump from i-Fax

Figure 39: More Debug Data from i-Fax: a *vertical line*

4.4.2.1 Data Format Constructions and Considerations

The parameters are actually in Prolog predicate format, although several different styles are suitable. An earlier version of the program utilised an HTML-style extension syntax. A verbose, HTML-type text-based syntax is convenient for legibility and ease of understanding, as well as portability across different platforms. An alternative would be a concise binary bit-oriented or tokenised syntax, if data sizes or transfer rates are important. Such a representation is preferable for data compactness considerations, although not necessarily legibility, as it is not text-based.

In the implementation described in the thesis, Prolog was chosen as a convenient output syntax. This is because the objects are later collated for subsequent analysis by a Prolog engine. So the data output from one layer of the architecture can be fed directly into another layer of the implementation in this case, thus saving an intermediary, although somewhat trivial, translation phase.

In each case the predicate name (which must start with a letter in lower case) is followed by its first parameter, which is a unique *handle* identifying each object. In this case the handle is the lower case letter ‘h’ followed by an eight character unique and random string created by a standard ‘C’ library function.

Other parameters may follow depending on the object type, as detailed in the appendices.

4.4.2.2 Further object composition

Further successfully recognised and decomposed lines will be similarly spooled out to the output subdirectory. Images that are not successfully recognised are saved as vector traces, for instance, for later recomposition.

Upon completion of the composition of the fax, the user would select the **Fax/Send** menu choice. This causes the following to occur:

1) a wrapper header is generated and spooled to the master output file

2) the primitive object files are concatenated one by one onto the end of this file

See Appendix B for an example master output file.

4.4.2.3 Spooling Subdirectory Usage

In the current implementation the master output file is named *ifax.pro* (with a deliberate default *.PRO* Prolog extension*)* in a subdirectory named **\OUT**; primitive object files will have been spooled to the **\OUT\OBJECTS** subdirectory. Other implementations could have a direct interface between program modules, with data buffers being passed directly, instead of an intermediate file definition. However, it was felt to be more instructive throughout the development of all the elements of the software suite to retain the components as separate, highly-defined modules.

4.4.3 The Transporter, XPORT

A template transporter application, **XPORT.EXE**, demonstrates the application of an independent transport mechanism to convey the resultant master output file *ifax.pro* from one location to another. In practice, this transporter could take a number of forms, e.g.:

a) an encapsulator to wrap the file in SMTP, so as to be conveyed over an Internet email connection

b) an IPX, TCP/IP or other transport protocol driver

c) a modem dial-up driver

d) a native transport layer

Note that a post-processing compiler such as a tokeniser, compression and/or encryption engine could also be used to effect more efficient and faster transportation.

In this implementation the transport layer engine demonstrates environment independence by simply copying the file from one subdirectory location to another, i.e. from the subdirectory **C:\OUT** to the subdirectory **C:\IN**.

4.4.4 Interpreting and Viewing the External Representation

The transportable external representation, comprising a complete description of the input data in terms of (i) text elements (ii) informal object descriptions and (iii) any other data, can be further processed in several ways. At a Class 1 level, the component informal objects (such as

a set of rough straight lines) can be recomposed by a viewer program into what should be a CI-equivalent representation of the original object, such as a sketch of a house.

At a Class 2 level, the set of informal objects can be further processed to glean instances of higher-order informal objects. For instance, a set of four rough straight lines, if correctly *constrained* with the correct attributes (lengths and angles of attachment, for instance), might be deemed by a recognition engine to constitute a *square*. A more complex set of objects might similarly be deemed to constitute a *house*. This new higher-level representation could then be passed to another layer (e.g. a transportation function) as a single informal object – *House* for instance. Part of this object definition would also be a measure of the level of *informality.*

A high-level viewer would then be able to take as input the high-level representation, and would reconstitute it in its own interpretation. If all goes well, and the gist retention is sufficient, the resultant graphical output should be a cognitively informally identical image – e.g. a house recognisable to the original creator as essentially the same one as in the original intention.

4.4.5 DOSView - the Simple DOS Graphical Viewer

DOSView is a simple informal interface object viewer, which creates a recomposition using independently developed algorithms; the program is a DOS program, as opposed to a Windows one, with no common algorithm code base shared with i-Fax or other modules. (Indeed, the program was deliberately developed in 'C' as opposed to C++, and for the DOS operating system and not for Windows, to emphasise this independence from the original codebase of C++).

DOSView creates a template fax output screen, displaying its built-in layout (which may of course be different than the layout of the sending program), and then fills in textual information parsed from the input file as necessary. The vector trace graphics images are subsequently displayed, as are the program's own interpretations of informal objects. This is illustrated in Figure 40, which is DOSView's interpretation of the external representations of the actual sketch of a house as generated by i-Fax, illustrated previously in Figure 36.

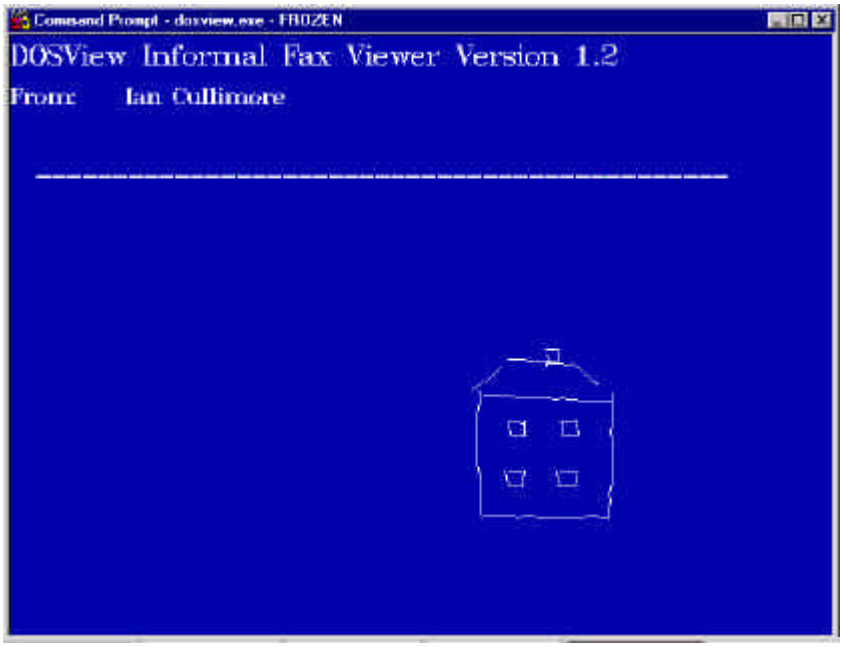

Figure 40: DOSView

4.4.6 The Prolog Object Recogniser Engine, EXAMINER

The informal object decomposition engine of the i-Fax program is a Class 1 model architecture, having embedded understanding of only a single primitive informal object and associated informal cognitive dimensions, namely the rough straight line (RSL). An integrated decomposition engine could be constructed with a higher level of understanding (Class 2 or

Class 3); instead, in this implementation a secondary and distinct Class 2 recognition engine is *Examiner*. This is a Prolog intelligence engine developed using Cogent Prolog from Amzi! Inc. (Amzi, 1995). Figure 41 shows the source code for Examiner, *examiner.pro*, loaded into the Integrated Development Environment.

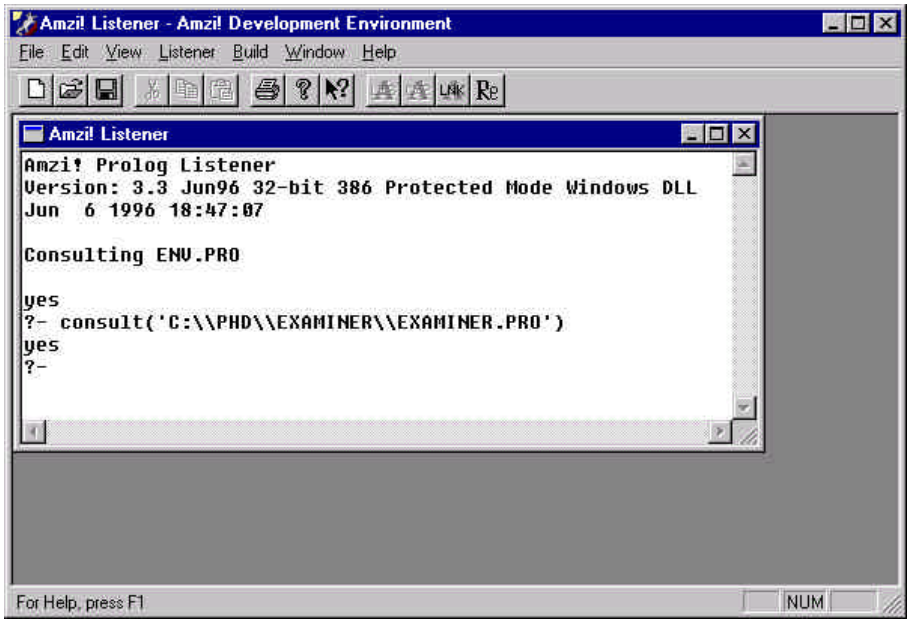

Figure 41: Examiner, written in Prolog

Examiner takes as its input the ASCII text file named ifax.pro which was previously the output of the i-Fax program. This file consists of, amongst other data, Prolog predicates (as detailed earlier) which describe the characteristics of informal objects.

Examiner seeks, in Prolog style, to recognise higher levels of object abstraction according to its in-built rules. For instance, it will seek to recognise level two primitive objects such as a *square*. A square is defined, in Prolog rule fashion, as four straight lines of equal length joined successively one to the other at right angles. Similarly, rectangles, triangles and other

geometrical shapes can be recognised. Much use is made of a *joined_to* rule, detailed in the source code listings in Appendix A.

This is the current level of development of Examiner. But by making use of informal cognitive notions such as *within, on_top_of, to_the_left_of* and so forth more complex level three objects such as a house could be recognised, for a house is composed primarily of a number of rectangles:

1. a main rectangle

2. with four smaller squares within it

3. one to the top left

4. one to the bottom left

5. one to the top right

6. one to the bottom right

7. a small vertical rectangle in the bottom centre within the main rectangle

8. two inward sloping lines from the top left and top right of the main rectangle...

and so forth.

4.4.7 The Intelligent Viewer, i-View

DOSView understands only the basic primitive of the rough straight line (RSL) and its associated dimensions. That is, it deals only at the Class 1 level. However, with the

development of a more complex decomposition and recognition engine, and hence a more complex set of informal objects, it is desirable to also be able to use a more sophisticated viewer, i.e. one that operates at a level higher than Class 1.

i-View takes high-level informal objects and re-interprets and displays them according to its own built-in rules for understanding. For instance, i-View takes the input of the informal object *House* and constructs an image in its own understanding of a cognitively informally identical house, as shown in Figure 42.

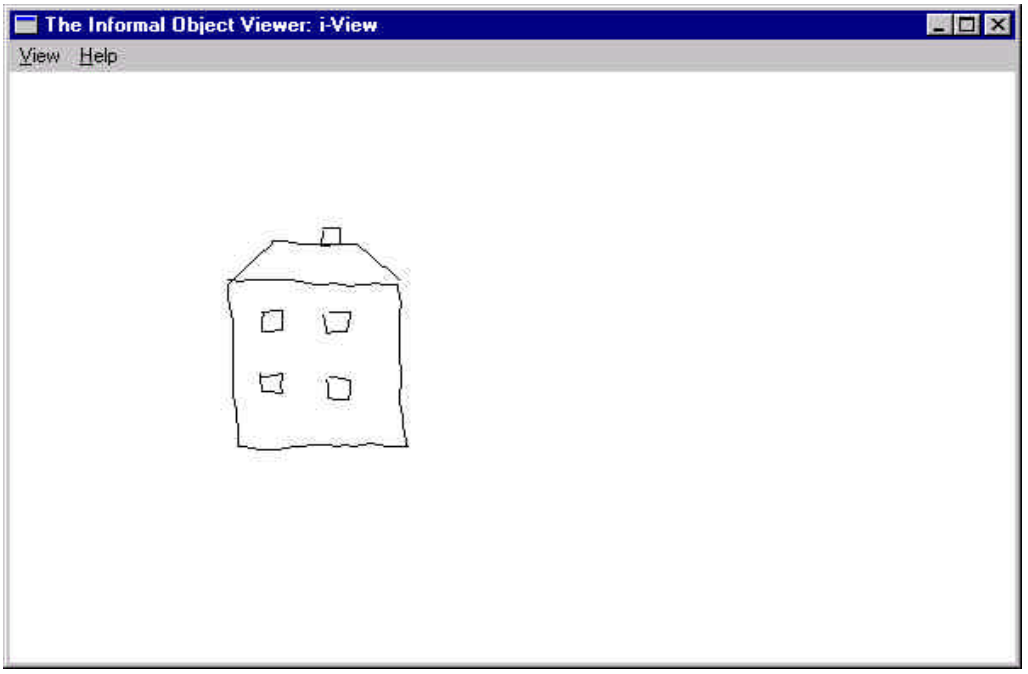

Figure 42: i-View's interpretation of a *House*, as originally output from i-Fax

The software program i-View has a primitive viewpoint of houses. Its rendition of a house is a very simple one – two dimensional view from the front, four windows, no door, side slopes to a flat roof, and a chimney on the right. This is exactly the same as the gist of a house as understood by i-Fax. A more sophisticated scenario would entail the high-level informal object

for the house having a number of attached informal cognitive dimensions, such as <with door>, <two chimneys>, <one window downstairs> and so forth.

i-View's informal cognitive dimensions are currently built-in and hard-coded, but they can be changed through a recompilation of the source code.

Figure 43 shows how i-View interprets its rendition of this house with no informality applied. Note how it is a boundary condition – a "perfect" rendition of its understanding of a "house". That is, all the lines are perfectly straight, with windows placed symmetrically and so forth. Of course, this would be a good start for an architect's or builder's plans for a well-engineered house, but may not be suitable as a "rough sketch" of an idea for a house design to stimulate further thought and design.

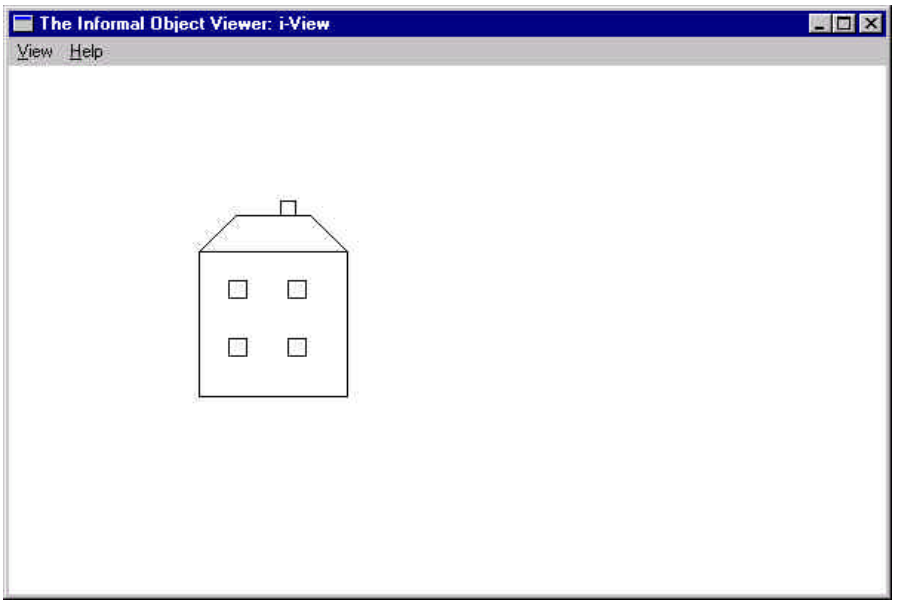

Figure 43: i-View's rendition of a *House* without informality applied

4.5 An Example Scenario

As an example, consider the case of a user who needs to send directions in some way to someone else as to how to get to the user's office or home. As discussed earlier in the thesis, the user has a number of options, for instance:

1) photocopy a map and send that through the post

2) photocopy a map and fax it

3) scan in the map and attach that to an email to send to the other person

4) draw in a map on paper, and fax or scan and then email that

5) use a drawing package on a computer to create a line drawing map, and then print that out for subsequent faxing, or email the graphical image

and so forth.

Sending a map by email or fax has the obvious advantage of speed, as the conventional postal system takes at least a day to deliver.

As part of the research for this thesis, several examples were collected of the way in which different people chose to accomplish the above. This was done over the course of three years, while working in the computer industry and observing the way that this task was undertaken in real-life situations. The examples monitored were generally the result of a request to another company to supply a map of how to find its offices, for an upcoming meeting for instance. In the majority of examples (14 out of 19) users chose to draw a sketch of a map by hand for

subsequent faxing. In all of these cases, the sender and recipient would have had access to computer and electronic mail. In a minority of cases the users chose to attempt to draw a text representation in the body of an email message, or attach a previously designed document to an email.

In the scenario of using an informal interface system as described in this thesis to accomplish this task, the user would use the program i-Fax to sketch in the map. A combination of i-Fax and EXAMINER would then decompose the sketched map into the elements of:

1) rough straight line objects, and other primitives understood by i-Fax

2) higher-level primitives understood by EXAMINER (e.g. squares, circles, blobs, and other geometrical shapes, constraints such as attachment, etc.)

3) other elements not handled by i-Fax and EXAMINER, such as handwriting text and complex graphical images

Note that the current implementations of the software can handle the more primitive of these objects, but not all of them. In principle the software can be further extended to handle all of these cases.

The set of software would finally create an output file containing representations of all these types of objects. This file would then be sent by electronic mail to the recipient. The recipient would then run the i-View software to view the image.

Due to the variability of output of an informal interface system as described in this thesis, the reconstructed bitmap image of the map might not be exactly the same as the one the user had

originally created. However, if the system had worked successfully, the resultant image would be CI-equivalent to the original, and so the gist would have been retained and so the map would still portray the correct information. It is unlikely that the user would be able to create *exactly* the same image on subsequent attempts, although it should still be a CI-equivalent image and should contain the same gist. The recipient should be able to navigate successfully using the received map.

4.5.1 A Metric for Success

A metric for success for such an informal interface system would be if the map was reproduced successfully, i.e. containing the same gist, and CI-equivalent or cognitively equivalent.

This metric could be measured in a number of ways:

1) The recipient was able to navigate successfully using the map

2) An analysis was made of the map, comparing it to the original. All key elements would be examined for contextual correctness. For instance, the maps should be topographically equivalent.

3) The map could be compared to a formal version, e.g. an Ordnance Survey map, to verify that all key elements were correct

4) The resultant map could be shown to the original sender. If this person declared that the map was, in their opinion either *exactly the same* (i.e. CI-equivalent), or *similar but still*

conveying the same information (i.e. cognitively equivalent), then the gist would have been retained and the system would be deemed successful.

4.6 An Evaluation of the Informal Interface

This section provides an overall evaluation of the thesis' implementation of the informal interface, and details user studies that have taken place. The section also considers side-effects from using informality in user interface design. For instance, there can be a positive effect from data compression, since only the compact representation of *gist* needs to be physically transmitted. Distilling representations down to their gist may allow for efficient indexing systems. Also, informal interface representations are sometime dependent on locale. This may have a positive or negative effect on the system.

4.6.1 Goals of the Evaluation Study

There were a number of goals for the evaluation study. First, it was an experiment to see how well people were able to use the user interfaces of the software tools. It was also a study of how well the test subjects were able to draw lines and shapes using the available input tools (a mouse and a graphics tablet). These hand-drawn shapes (e.g. squares) were then presented as input to the High Level Recogniser software component, so its performance was also evaluated. The quality of the subsequent output (renditions of the gist of the shapes recognised) was then evaluated. Finally, the test subjects' reactions to the subsequent outputs, and the overall performance of the set of software tools, was evaluated.

From a design point of view, the evaluation study had benefits in that it was valuable feedback as to how well (or otherwise) the software components performed, and how users were able to interface with these components. It was also beneficial to gain the users' reactions as to

what types of useful systems might be able to be constructed out of such components, and therefore where future research directions might lie.

4.6.2 The User Studies

Section 1.5 detailed two specific ways in which a metric of the effectiveness of an informal interface or system can be measured. This chapter evaluates the elements of the implementation of the informal interface software system described in this chapter.

There are two proposed metrics for conducting an evaluation: 1) user appreciation studies with a sample of end users, and 2) a feedback loop system, in which an output from an informal interface system is fed back into itself. The first metric was used in this study. The software programs developed to demonstrate the essential elements of this thesis have proven to be of use, but are still at a relatively early stage of development. Later versions will become more sophisticated. Nevertheless, it was felt to be useful to conduct at least some preliminary user studies using the software developed to date.

This chapter summarises a small user study that was carried out as an initial evaluation of the implementation programs. Five users were selected, and some time was spent individually with them on a one-to-one basis. Each individual user was seated alone with the writer of this thesis in a room, in front of a computer that was running the implementation programs. Each user study occupied a period of time of about thirty minutes. The user's actions and comments were noted, and appropriate feedback and guidance given whenever necessary.

4.6.3 An Example Scenario, and the Results of User Studies

Supposing we want to "fax" a simple drawing to someone else, perhaps the sketch of our house, or a map of how to get to a certain place. As a simple example perhaps we "fax" just a rough-drawn object, such as a square. A simple square will serve as an example, as it illustrates how more complex objects (e.g. a house, which is made up of straight lines, squares, rectangles, sloping lines, etc.) could also be treated.

The five users were asked to draw by hand a rough sketch-like square into i–Fax. The users ranged in age from 5 to 54 years old, and were a mixture of male and female. Only one was particularly computer-literate. No particular explanation was made to the users of the underlying way that the set of software programs worked.

The users were first allowed to experiment with using either a mouse or a graphics tablet as the input device. Lines are drawn in i-Fax by first positioning the mouse cursor in the desired place, then dragging the mouse (or moving the stylus) with the left button held down. After some experimentation (approximately five minutes) the users were all able to draw reasonable straight horizontal and vertical lines, which would be successfully recognised by i-Fax as in Figure 38 and Figure 39. In one case, the value of one of the default informal cognitive dimensions (the "shakiness") was changed to increase the success rate of line recognition. (This user presumably had a more shaky hand for writing than those of the other subjects.) As would be expected, most of the users found some difficulty at first in drawing straight lines successfully, especially using the mouse. The graphics tablet was found to be better for drawing straight lines on their own, as it was more like using a conventional pen. However, users generally had more difficulty in drawing lines that joined up successfully to create the

corner of the square when using the tablet, than when using the mouse. They generally found it easier to make the join using the mouse, as the mouse cursor could first be positioned in the correct place, and then the line drawn. One particular rendition of a hand-drawn square is depicted in Figure 44.

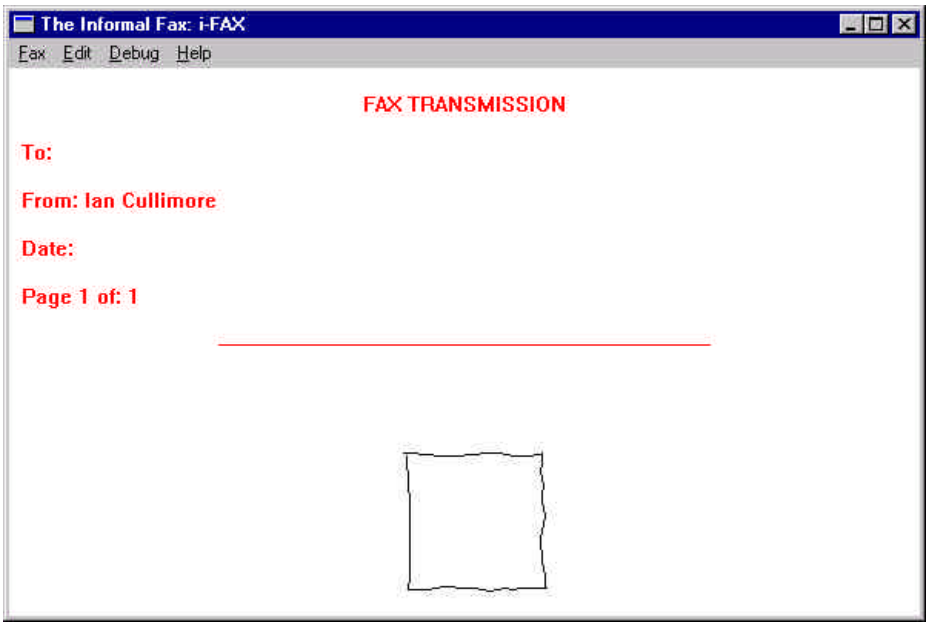

Figure 44: A human hand-drawn square entered into i-Fax

In one such typical trial, i-Fax generated the following set of four "rough straight line" decompositions:

```
objecttype(hBBBICOAA, line).
shakiness(hBBBICOAA,2).
period(hBBBICOAA,4).
direction(hBBBICOAA,90).
thickness(hBBBICOAA,3).
harmony(hBBBICOAA,0).
accuracy(hBBBICOAA,5).
length(hBBBICOAA,89).
startpoint(hBBBICOAA,256,256).

objecttype(hBBBICOAB, line).
shakiness(hBBBICOAB,2).
period(hBBBICOAB,4).
direction(hBBBICOAB,180).
```

```
thickness(hBBBICOAB,3).
harmony(hBBBICOAB,0).
accuracy(hBBBICOAB,5).
length(hBBBICOAB,92).
startpoint(hBBBICOAB,346,256).

objecttype(hBBBICOAC, line).
shakiness(hBBBICOAC,2).
period(hBBBICOAC,4).
direction(hBBBICOAC,270).
thickness(hBBBICOAC,3).
harmony(hBBBICOAC,0).
accuracy(hBBBICOAC,5).
length(hBBBICOAC,91).
startpoint(hBBBICOAC,349,346).

objecttype(hBBBICOAD, line).
shakiness(hBBBICOAD,2).
period(hBBBICOAD,4).
direction(hBBBICOAD,0).
thickness(hBBBICOAD,3).
harmony(hBBBICOAD,0).
accuracy(hBBBICOAD,5).
length(hBBBICOAD,88).
startpoint(hBBBICOAD,259,347).
```

Each definition set contains the “objecttype” (i.e. “line” for straight line), and the starting point co-ordinate position “startpoint” (where the origin is the top left hand corner of the i-Fax pane). Each definition also contains a set of informal cognitive dimensions describing the characteristics of the line. In this implementation, those dimensions logged are *shakiness, period, thickness, harmony, accuracy,* and *length.* Although harmony is included in the list of dimensions, it is not utilised in this implementation. The *direction* of the line is measured in degrees clockwise from the vertical, so there are four lines of directions *90, 180, 270* and *0 (*i.e. *360)* degrees.

Although all the users, after some practice, were able to successfully draw straight lines, it was found to be much more difficult to draw a set of four such lines that EXAMINER would recognise as a square. That is, the four lines might look reasonably square-like to the user, but

EXAMINER would find either (a) a number of disjointed lines, (b) often at best a rectangle, or (c) at best, and very occasionally, a square.

In this successful case, i-View then took the Class 2 informal object *Square* (which was spooled into an output file by Examiner) as input, and generated its rendition of the object as shown in Figure 45.

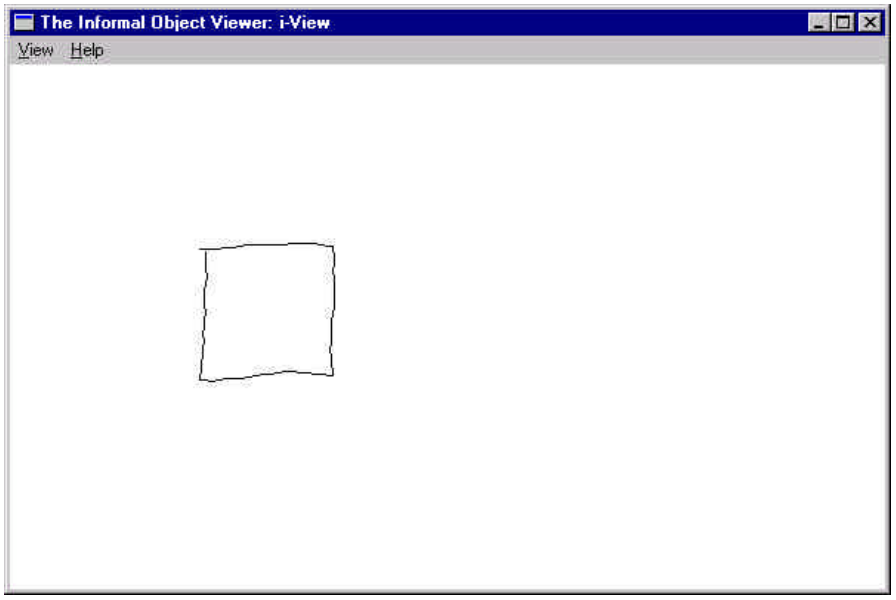

Figure 45: The resultant square, regenerated by i-View

4.6.3.1 A Metric of Success Drawn from the User Studies

In all cases, the users were able after some time to successfully draw four rough straight lines that were subsequently determined to be a square by EXAMINER.

These squares were then subsequently reconstituted by i-View, and viewed by the relevant user who had created the original square. The process of input, examination and subsequent

reconstitution generally took three or four minutes. The user was always shown the reconstituted square, and not the original one.

The users were then asked:

(i) "is this rendition a square?"

and if the answer was "yes", the question was asked:

(ii) "is the square exactly the same one as the one that you originally drew?"

and if the answer was "no", the question was asked:

(iii) "is it to all intents and purposes the same one?"

and if the answer was "no", the question was asked:

(iv) "why not?"

or if the answer was "yes" to (iii), the question was asked:

(v) "why don't you think that the square is exactly the same?"

The five users successfully created a total of 12 squares, all of which were successfully recognised by EXAMINER as squares, and subsequently reproduced as renditions of squares by i-View. When asked question (i), in all cases the users accepted that the resultant images represented squares. In four cases the users answered "yes" to question (ii), and in six cases, having answered "no" to (ii), they answered "yes" to (iii). In this case, being questioned further by (v), they claimed that they could remember that the lines were not exactly the same as the ones that they had originally drawn. In the cases that the users answered "yes" to (ii), they

stated that they thought that the squares were indeed exactly the same ones as the ones originally drawn.

In two cases users answered "no" to question (iii). When questioned further by (iv), in both cases they stated that the lines of the square appeared unrecognisable from the ones that they had originally drawn, although they accepted that the resultant images were reasonable squares.

Users who had answered "yes" to (iii) were further questioned about the nature of the squares subsequently reproduced by i-View. In all cases they accepted that the i-View squares were satisfactory renditions of a square, and therefore conveyed the same meaning as their original drawing.

4.6.3.2 Conclusions Drawn from the User Studies

1) The mouse and stylus were found to awkward devices to use, especially for drawing straight lines. Other types of input devices could be tried.

2) The users took some time to get used to the system, and had to experiment for a few minutes to be able to create successful images (i.e. "squares" that were recognised as such by EXAMINER).

3) The resultant squares were in most cases thought to be either exactly the same, or to all intents and purposes the same, as the originals.

4) The case of using squares, although illustrative, is simplistic, and more complex diagrams need to be examined in the future.

4.6.3.3 Limitations of the User Studies

Since the current implementations of the software programs are rather limited in their capabilities, notably EXAMINER which, while being able to recognise *squares* cannot yet recognise a *house*, the user studies are useful, in that they provide some preliminary feedback, but not exhaustive. It is hoped to be able to develop the software programs further in the future, and once more complex shapes such as a *house* and diagrams of *maps* can be recognised, more user studies can take place.

4.6.4 Side Effects of Utilising Informal Interfaces

There are a number of interesting side-effects which come out of the representations used in the type of informal interface detailed in this thesis, as discussed in this section.

4.6.4.1 Data compression and speed of transmission

It will have been seen that a complex image such as a sketch of a house might occupy perhaps a pixel bit-map of 200 * 200 bits or approximately 5 kilobytes of memory – and that is at a low resolution in monochrome. Scanners and fax machines operate at much higher resolutions, and hence create bit-map files of even greater size - even more so when colour is involved – although of course such images are compressed and encoded for transmission.

A more compact representation for informal objects is straight-forward to devise. The representation used in the implementation described in this thesis is deliberately verbose, and yet at its ultimate a complex image like a hand-drawn sketch of a house can be distilled down to a single token called *house*. At the worst this is five bytes of ASCII characters (admittedly

within some file or packet structure), and at the best could be represented as a token of perhaps just a few bits. Of course this representation is, as always, context-dependent, and may require supplementary objects. Nevertheless, it is felt that such informal representations would generally provide for very compact encapsulations.

In this implementation, a square produces an output file from i-Fax that is 1,258 bytes in size. EXAMINER's output rendition is even smaller, at only a few dozen bytes. A similar sized monochrome bitmap image of such a square is approximately 2,500 bytes in size. A side effect of this is speed of transmission. Fewer bytes of data, especially compared to the large amount of data generally involved in Internet transmissions, mean quicker transmission times over slow telecommunications links.

This adds value to the design and implementation of object-based systems, such as the informal interface described in this thesis, because of the inherently compact representations of gist that may result. In this case, this has the potential benefit of speeding up data transmission, and so making a real-life implementation of such a system practical for some situations. For instance, mobile data rates over GSM are currently limited to 9,600 baud, as compared to perhaps several hundred thousand baud over fixed lines.

4.6.4.2 Localisation and Locale Information

There is an interesting by-product that can be utilised in reconstruction techniques - that of taking account of the locale of the user (which can typically be determined from the underlying operating system). It may be advantageous to sometimes offer a rendition of reconstructed informal objects in a style sympathetic to the location and culture in question. For instance, the

rendition of an image of a mail box could be slightly different (but perhaps more familiar) depending on whether it is for an audience in France, the United States, Britain, or wherever.

4.6.4.3 Indexing by Gist

The type of informal interface system described here lends itself to an indexing system. Since representations, such as sketch-like houses, are distilled down to a single or small number of informal objects representing their *gist*, they are a compact representation and suitable for cataloguing. Representations can be indexed according to any given particular level of informality. So, at the most "loose" level of definition (i.e. at a high level of informality), all representations of sketches of various different types of houses, for instance, are afforded the simple informal object definition of *House*. Applying a lower level of informality would yield finer detail for the representations, thus sub-categorising them. For instance, houses could be sub-indexed according to *StyleOfRoof*, *NumberOfChimneys*, and so forth.

Again, this adds value to the design and implementation of object-based systems, such as the informal interface described in this thesis, because of the alternative way that such objects could be represented internally, and subsequently catalogued and indexed. Such an object-based system lends itself well to utilising *inheritance*, with derived objects adding more detail to their structure and representation as required through their member functions and objects.

4.7 Summary of this Chapter

This chapter describes a set of software tools and applications developed to illustrate the implementation of one type of informal interface, as described in this thesis. The components of the system are (i) an Informal Interface Object Browser (*I2OB*), which is a tool for

investigating informal cognitive dimensions and shapes; (ii) *i-Fax*, which is a "fax program" style application which recognises hand-drawn rough straight lines; (iii) a transport layer exporter function *XPORT*, which conveys the external representation from one client computer to another; (iv) *EXAMINER*, an AI-based recognition engine which attempts to recognise higher-level constructs (such as a *Square*) from lower level constructs (i.e. rough straight lines); (v) *DOSView*, a low-level informal object viewer, which reconstructs rough straight lines from their informal object representations; and (vi) *i-View*, a high-level informal object viewer, which reconstructs high-level informal object representations such as a *House* and a *Square*.

The chapter demonstrates an example scenario of using an informal interface system, whereby a user enters a hand-drawn map into i-Fax, which is then processed by a Low Level Recogniser into its fundamental informal objects (rough straight lines, rough curves, etc.). These low-level representations are further processed by a High Level Recogniser. This high-level representation can then be transmitted to another distant user, to be reconstructed into an informal rendition of the original sketch, which should be judged to be CI-equivalent to the original map.

The chapter also considers metrics for evaluating the success of such a system, and gives the results of one such evaluation, being user studies. A sample of users (generally non-computer literate) were asked to work through the process of drawing in a free-hand sketch of a square into one of the software programs. This was subsequently recognised as (a) four rough straight lines, and then, after a further recognition stage, (b) an informal square. A further software

program then regenerated its own rendition of the square from the simple informal object "square", to the general satisfaction of the users.

The chapter also considers some potential benefits from side effects of implementing an informal interface system as described in this thesis. Informal object representations should generally be smaller, and faster to transmit, than conventional representations. Also, informal objects are potentially locale-dependent, which may be used to advantage. They may also provide for efficient and compact ways of indexing some types of information.

5. Conclusions and Further Research Work Directions

5.1 Introduction

The methods described in this thesis provide an approach for designing and constructing computer interfaces and software programs utilising notions of informality. This approach has been demonstrated at two levels: first, utilising sketch input and output at an immediate interactivity level, and second, the notion of retaining the underlying informal internal representation across manipulation operations and transport interfaces. The flow of input, deconstruction, representation, manipulation, reconstruction and output has been examined, and the concepts of sketch as an interface and interaction design paradigm discussed.

To substantiate the claims of the thesis a software implementation has been described and developed in the form of software tools and applications, in the guise of an email or fax system utilising informal architectures. The architecture's platform and implementation has been demonstrated. User studies have been conducted, and the results set out and examined.

5.2 Principal Contributions of the Thesis

The thesis has introduced the notion of employing informality in Human-Computer Interaction. Specifically, it has introduced the notion of using sketch input and output, and allowing for tolerance of input and variability of output. It has also introduced the notion of the *gist*, i.e. the fundamental essence, of a representation.

The thesis has also described an architecture for the structure and analysis of such informal representations and systems, and a taxonomy of classes for such systems. An example

system's implementation in software is described in detail to help support and illustrate the claims of the thesis. User studies are also presented to help support and illustrate the aims of the thesis, and to help highlight potential areas of further research, as well as further user studies.

5.3 Further Research Topics

More sophisticated or different domain-specific taxonomies of informal objects could be construed than have been considered so far in this thesis. For instance, in the implementations described in this thesis the basic construction has been around the Rough Straight Line, and the simple constructs (such as squares and triangles) arising out of it. There are two directions in which this particular line of research could go: (1) more sophisticated systems could be designed and developed to extend the implementations of the constructs (into more complex objects such as *houses*, *letter boxes*, *cars* and so forth), and (2) new classes of constructs could be devised from other underlying primitive objects - perhaps using a *circle* or other geometric shape as a fundamental primitive building block.

The implementation has been demonstrated in a mixture of 'C', C++ and Prolog, but something like a dedicated scripting language could be developed. There are also other domains, applications and problem areas to which these concepts could be applied in terms of future implementations.

5.4 Informal Interface Futures

Although, for the purposes of this research, conventional input devices such as the mouse and the keyboards were assumed to be the input devices being used, it is interesting to conjecture

what sorts of input devices might be used in the future: holograms or Virtual Reality helmets, perhaps.

Using just a conventional mouse and a keyboard has created some limitations in the usability of the implementation software. Informal input in the guise of sketching is, by definition, well suited to the use of a stylus, as this is the natural way that people sketch using pen on paper. Drawing with a mouse is a more difficult task, as has been illustrated in the user studies. However, over the course of performing the research for this thesis, stylus-based computers such as the 3Com Palm have been enjoying great popularity. It may be that they are well suited to being utilised in implementations in the future.

5.4.1 An Informal Interface Display

One could imagine that an informal interface display device could be constructed, operating as a client in a similar manner to the X-Windows or other client-server graphics device system. An informal interface display is suited to informal interface representation structures, as the reconstruction display engine could be embedded within the display device. So, rather than have a client software driver reconstruct an informal sketch output image for display on a conventional bit- mapped raster graphics display, the display itself could accept informal object definitions.

5.5 Future Development Directions

It will be gratifying if this research into informal interfaces also spurs on development work into applicable areas; such work has already been progressing in areas such as Pad++ (Meyer & Crumpton, 1996) and the Electronic Napkin (Gross, 1996).

The benefits and rationale behind utilising informal interface concepts in HCI might be twofold. First, this thesis has attempted to show that such a sketch-based informal interaction may be beneficial for the end user, both in providing a natural and familiar way in which to operate, and also in providing for a useful framework for the representation of data and ideas. Secondly, there may be spin-offs from the use of informal interface architectures as described in this thesis, such as speed of data transfer and compression of data. Informal objects can be designed to be inherently small and concise, and so are quick to transmit over a slow transport medium (such as a mobile telephone data connection). Since the representations are compacted and condensed, and in a sense mangled, there may be scope for inherent encryption too.

As has been stated, this fundamental notion of applying informality to HCI is twofold, as both the overlying interface design, and the underlying computer operation and internal representations must be considered. It is not necessary for informality to be applied at both these high and low levels. A partially informal system could apply them at just, say, the higher level of direct human-computer interaction, in the form of sketch input from the human user, and sketchy-style output from the computer. Such a partially informal system could operate at just this superficially informal level, with the underlying representations and operations actually being modelled on conventional operating systems, object systems, data representations and architectures.

Conversely, some conventional-looking computer systems, with conventional input and output mechanisms, could utilise underlying notions of informality in some of their internal

representations. For instance, it may be useful to decompose external representations into fundamental internal primitives that encapsulate the gist of the representation.

Finally, a fully-informal interface system, as according to this thesis, could use both of these elements, and utilise both the higher level of informal input and output in the human-computer interaction cycles, as well as low-level informal internal representations.

6. Bibliography

Amzi (1995). *Amzi! Prolog Users Guide and Reference, Version 3.3.* Amzi! Inc., Stow, MA USA. http://www.amzi.com

Banahan, M.F. & Rutter, A. (1982). *Unix – the Book*. Sigma Technical Press.

Barfield, L., van Burgsteden, W., Lanfermeijer, R., Mulder, B., Ossewold, J., Rijken, D. & Wegner, P. (1994). Interaction Design at the Utrecht School of the Arts: Interaction Design Education. *SIGCHI Bulletin*, Vol. 26, No. 3, July 1994. p 69.

Booch, G. (1994). *Object-Oriented Analysis and Design*. (Second edition). Benjamin/Cummings Publishing Company, Inc.

Borland (1992). *Borland C++ Version 4.5*. Borland International, Inc., Scotts Valley, CA USA.

Borning, A.H. (1979). *Thinglab - A Constraint-Oriented Simulation Laboratory*. Xerox PARC paper and Stanford Computer Science Department Report STAN-CS-79-746.

Borning, A.H. & Duisberg, R.A. (1986) Constraint-Based Tools for Building User Interfaces. *ACM Transactions on Graphics*, Vol. 5, No. 4, pp 345-374.

Brocklehurst, E.R. (1989). The NPL electronic paper project. *International Journal of Man-Machine Studies (1991), **34**, pp 69-95.*

Bundy, A. (1977). *Will it Reach the Top? Prediction in the Mechanical World.* D.A.I. Research Report No. 31.

Carroll, J.M. (1991). *Designing Interaction. Psychology at the Human-Computer Interface*. Cambridge University Press.

Chattopadhyay, S. & Das, P.P. (1991). A new method of analysis for discrete straight lines. *Pattern Recognition Letters*, 12 (1991) pp 747-755. Elsevier.

Citrin, W. & Gross, M. (1996). *Distributed Architectures for Pen-Based Input and Diagram Recognition*. ACM Workshop on Advanced Visual Interfaces (AVI '96).

Clocksin, W.F. & Mellish, C.S. (1981). *Programming in Prolog.* Springer-Verlag.

Cohn, A.G., Randell, D.A. & Cui, Z. (1993). Taxonomies of Logically Defined Qualitative Spatial Relationships. In Guarino, N. & Poli, R. (Eds) *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer.

Davis, R.C., Lin, J., Brotherton, J.A., Landay, J.A., Price, M.N. & Schilit, B.N. (1998). *A Framework for Sharing Handwritten Notes.* UIST '98, San Francisco, CA. ACM.

Dix, A., Finlay, J., Abowd, G. & Beale, R. (1993). *Human-Computer Interaction*. Prentice Hall

Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang., J & Welch, B. (1992). *Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration.* ACM Press.

Fish, R.S., Kraut, R.E. & Root, R.W. (1992). *Evaluating Video as a Technology for Informal Communication.* ACM.

Fish, J. & Scrivener, S.A.R. (1990). Amplifying the Mind's Eye: Sketching and Visual Cognition. *Leonardo*, Vol. 23, No. 1, pp 117-126.

Fisher, D.A. (1991). What is Informalism? In *Proceedings of the Workshop on Informal Computing*. Incremental Systems Corporation.

Foley, J.D., Wallace, V.L. & Chan, P. (1984). The Human Factors of Computer Graphics Interaction Techniques. In Preece, J. & Keller, L. (Eds.) *Human-Computer Interaction*. Prentice Hall.

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

Green, T.R.G. (1990). The Cognitive Dimension of Viscosity: a Sticky Problem for HCI. In Diaper, D., Gilmore, G., Cockton, G. & Shackel, B. (Eds.), *INTERACT '90*. Elsevier Publications.

Green, T.R.G. (1991a). Comprehending and Manipulating Complex Information Structures. *Theory & Methodology of Cognitive Science Applied to HCI Problems.* Queen Mary & Westfield College, London.

Green, T.R.G. (1991b). Describing Information Artefacts with Cognitive Dimensions and Structure Maps. In Diaper, D. & Hammond, N.V. (Eds.), *HCI '91: Usability Now*. Cambridge University Press.

Green ,T.R.G. & Blackwell, A.F. (1996). *Thinking about Visual Programs.* Presented at Thinking with Diagrams. Colloquium of IEE Computing and Control Division, Digest No. 96/010, 5/1-5/4.

Gross, M.D. (1996). *The Electronic Cocktail Napkin – computer support for working with diagrams.* Design Studies, Vol. 17, No. 1, pp 53-69.

Gross, M.D. & Do, E.Y-L. (1996a). *Ambiguous Intentions: a Paper-Like Interface for Creative Designs.* UIST '96, Seattle WA. ACM.

Gross, M.D. & Do, E.Y-L. (1996b). *The Electronic Cocktail Napkin Project*. University of Colorado. http://wallstreet.colorado.edu/napkin

Gusgen, H.W. (1989). *CONSAT: A System for Constraint Satisfaction*. Pitman Publishing.

Henier, J.M., Hudson, S.E. & Tanaka, K. (1999). *Linking and Messaging from Real Paper in the Paper PDA*. UIST '99, Asheville, NC. ACM.

Hollan, J. & Stornetta, S. (1992). *Beyond Being There.* ACM Press. Johnson, J., Roberts, T.L., Verplank, W., Smith, D.C., Irby, C.H., Beard M. & Mackey, K. (1989). The Xerox Star: A Retrospective. In Baecker, R.M., Grudin, J., Buxton, W.A.S. & Greenberg, S. (Eds.) *Human-Computer Interaction: Toward the Year 2000* (second edition). Morgan Kaufmann Publishers, Inc., 1995.

Igarashi, T., Matsuoka, S. & Tanaka, H. (1999). *A Sketching Interface for 3D Freeform Design*. SIGGRAPH '99, Los Angeles. ACM.

Kass, M. (1992). CONDOR: Constraint-Based Dataflow. *Computer Graphics*, 26, 2, July 1992. ACM Press.

Kernighan, B.W. & Ritchie, D.M. (1978). *The C Programming Language*. Prentice-Hall, Inc.

Kuipers, B.J. (1975). A Frame for Frames: Representing Knowledge for Recognition. In Bobrow, D.G. & Collins, A.M. (Eds) *Representation and Understanding*. Academic Press.

Lansdown, J. (1985). Not Only Computing - Also Art. *Computer Bulletin*. Series III, 1 (Part 2), 18-19.

Leclerc, Y.G. & Fischler, M.A. (1992). An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3D Wire Frames. *International Journal of Computer Vision*, 9:2, 113-136. Kluwer Academic Publishers.

Leler, W. (1988). *Constraint Programming Languages: Their Specification and Generation*. Addison-Wesley.

Lewis, S.B., Mateas, M., Palmiter, S. & Lynch, G. (1996). Ethnographic Data for Product Development. *Interactions*, Vol. III, No. 6, November + December 1996. p 61. ACM.

Lohse, J.A. (1991). A Cognitive Model for the Perception and Understanding of Graphs. *CHI '91 Proceedings*. ACM Press.

Long, Jr., A.C., Landay, J.A., Rowe, L.A. & Michiels, J. (2000). *Visual Similarity of Pen Gestures*. CHI 2000, The Hague, Amsterdam. ACM.

Maguire, M.C. (1985). A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human-Computer Interfaces. In Preece, J. & Keller, L. (Eds.) *Human-Computer Interaction*. Prentice Hall.

Marr, D. (1982). *Vision*. W.H. Freeman and Company.

Marill, T. (1989). Emulating the Human Interpretation of Line-Drawings as Three-Dimensional Objects. *International Journal of Computer Vision, 6:2, 147-161*. Kluwer Academic Publishers.

McCorduck, P. (1990). *Aaron's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen*. W.H. Freeman and Company.

Meyer, J. (1996). *EtchaPad - Disposable Sketch Based Interfaces*. ACM SIGCHI '96 short paper category.

Meyer, J. & Crumpton, M. (1996). *Creating Informal Looking Interfaces*. NYU Media Research Laboratory.

Microsoft Corporation. (2000). *Microsoft Visio*. http://www.microsoft.com/office/visio

Minsky, M. (1975). A Framework for Representing Knowledge. In Collins, A., & Smith, E.E. (Eds.) *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, Morgan Kaufmann Publishers, Inc. pp156-189, 1988.

Montalvo, F.S. (1985). *Diagram Understanding: The Intersection of Computer Vision and Graphics*. Artificial Intelligence Laboratory, Massachusetts Institute of Technology. A.I. Memo 873.

Montalvo, F.S. (1990). *Acquiring and Validating Qualitative Visual Properties.* DEC Cambridge Research Lab.

Moran, T.P., van Melle, W. & Chiu, P. (1998). *Spatial Interpretation of Domain Objects Integrated into a Freeform Electronic Whiteboard.* UIST '98, San Francisco, CA. ACM.

Mundie, D.A. & Shultis, J.C. (1991). *Proceedings of the Workshop on Informal Computing*. Incremental Systems Corporation.

Negroponte, N. (1973). Recent Advances in Sketch Recognition. *Proceedings of 1973 National Computer Conference and Exposition (42), AFIPS '73, NewYork.* AFIPS Press. pp 663-675.

Negroponte, N. (1977). On Being Creative with Computer Aided Design. *Information Processing,* 77, pp 695-704. I.F.I.P. Amsterdam, North-Holland.

Negroponte, N. & Taggart, T. (1971). HUNCH - An Experiment in Sketch Recognition. *Computer Graphics*. W. Giloi, Ed. Berlif.

Norman, D. A. (1986). Cognitive Engineering. In Norman, D.A. & Draper, S.W. (Eds.), *User Centered System Design*. Lawrence Erlbaum Associates.

Norman, D. A. (1988). *The Psychology of Everyday Things*. Basic Books.

Pao, D.C.W., Li, H.F. & Jayakumar, R. (1992). Shapes Recognition Using the Straight Line Hough Transform: Theory and Generalization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 11.

Pedersen, E.R., McCall, K., Moran, T.P. & Halasz, F.G. (1993). *Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings.* ACM Press.

Perlin, K. (1985). An Image Synthesiser. *Computer Graphics*, Vol. 19, No. 3.

Preece, J. (1983). Graphs are not Straightforward. In Green, T.R.G., Payne, S.J. & van der Veer, G.C. (Eds) *The Psychology of Computer Use*. Academic Press.

Preece J. & Keller, L.S. (1990). (Eds). *Human-Computer Interaction*. Open University.
Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. & Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley Publishing Company.

Reeker, L. (1991). Informalism In Interfaces. *Workshop in Informalism, Santa Cruz, May 28-31 1991*. Incremental Systems Corp.

Reisberg, D. (1987). External Representations and the Advantages of Externalizing One's Thoughts. *Proceedings of the IX Annual Convention of the Cognitive Science Society, Seattle, WA.* Lawrence Erlbaum Associates.

Robertson, S.P., Zachary, W., & Black, J.B. (1990). *Cognition, Computing and Co-operation*. Ablex Publishing Corp.

Rogers, Y., Rutherford, A., & Bibby, P. (Eds.) (1992). *Models in the Mind: Theory, Perspective and Application*. Academic Press.

Sannella, M., Maloney, J., Freeman-Benson, B. & Borning, A.H. (1992). Multi-way versus One-way Constraints in User Interfaces: Experience with the Delta Blue Algorithm. In *Software - Practice and Experience*, Vol. 23(5), pp 529-566, May 1993.

Schilit, B.N., Golovchinsky, G. & Price, M.N. (1998). *Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations.* CHI '98, Los Angeles, CA. ACM.

Scrivener, S.A.R. & Clarke, S.M. (1994). Sketching in Collaborative Design. In *Interacting with Virtual Environments*, pp 95-118, John Wiley and Sons Ltd.

Scrivener, S.A.R., Clarke, S., Clarke, A., Connolly, S., Palmén, H. Smyth, M. & Schappo, A. (1994). Real-time Communication between Dispersed Work Groups via Speech and Drawing. In Scrivener (ed.) *Computer-Supported Co-operative Work*, pp 51-56, Avebury Technical, Ashgate Publishing Ltd.

Scrivener, S.A.R., Harris, D., Clarke, S.M., Rockoff, T. & Smyth, M. (1993). Designing at a Distance via Real-time Designer-to-designer Interaction. *Design Studies*, Vol. 14, No. 3, pp 261-282. Butterworth-Heinemann.

Sharples, M., Hogg, D., Hutchison, C., Torrance, S. & Young, S. (1989). *Computers & Thought: A Practical Introduction to Artificial Intelligence*. Bradford Books/MIT Press.

Stefik, M. (1981). Planning With Constraints. *Artificial Intelligence*, 16, 111-140.

Straforini, M., Coelho, C., Campani M., and Torre, V. (1992). The Recovery and Understanding of a Line Drawing from Indoor Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14., No. 2, February 1992.*

Sutherland, I.E. (1963). *Sketchpad: A Man-Machine Graphical Communication System*. Proceedings of the Spring Joint Computer Conference, Detroit, Michigan, May 1963, and Ph.D. thesis, MIT Lincoln Laboratory Report #296, January 1963, Cambridge, Mass.

Thimbleby, H. (1990). *User Interface Design.* ACM Press.

Verplank, W. (1989). Tutorial notes. In *Human Factors in Computing Systems, CHI '89.* ACM Press.

Whittaker, S., Frohlich, D. & Daly-Jones, O. (1992). *Informal Workplace Communication: What Is It Like And How Might We Support It?* ACM Press.

Wood, C.C. (1992). The Cognitive Dimensions of Sketches. In Wood, Davidge & Costa (Eds.) *The Fifth White House Papers: Graduate Research in the Cognitive & Computing Sciences in Sussex.* University of Sussex. CSRP 251.

Zhao, R. (1993). *Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors.* ACM Press.

Appendix A

Program Listings

Note that only partial listings are given for the software implementations, although all the essential elements of the code are presented here. Only extraneous functions not central to the code have been omitted for brevity.

I2OB Source Code Listing ('C')

```
/*
       The main body of the program
*/

LONG FAR PASCAL MainWndProc (HWND hWnd, WORD message, WORD wParam, LONG lParam)
{
  FARPROC lpProc;
  int i;

  switch (message) {
    case WM_COMMAND:
      switch( wParam )
         {
         case IDM_ABOUT:
         lpProc = MakeProcInstance ((FARPROC) About, hinst);
           DialogBox (hinst, "AboutBox", hWnd, (FARPROC) lpProc);
           FreeProcInstance (lpProc);
           return (DefWindowProc (hWnd, message, wParam, lParam));

         case IDM_HELP:
           lpProc = MakeProcInstance ((FARPROC) Help, hinst);
           DialogBox (hinst, "HelpBox", hWnd, (FARPROC) lpProc);
           FreeProcInstance (lpProc);
           return (DefWindowProc (hWnd, message, wParam, lParam));
```

```
case IDM_LINE:
  ClearWindow (hWnd);

  GetClientRect (hWnd, (RECT FAR*) &Rect);    /* Get size */
                                                /* of window */
  /* Work out the ideal location */
  if (iDirection <= 90) {
      StartX = (Rect.right - Rect.left)/3;
      StartY = ((Rect.bottom - Rect.top)/3)*2;
      } else {
        if (iDirection <= 180) {
           StartX = ((Rect.right - Rect.left)/3)*2;
           StartY = ((Rect.bottom - Rect.top)/3)*2;
      } else {
         if (iDirection <= 270) {
           StartX = ((Rect.right - Rect.left)/3)*2;
           StartY = (Rect.bottom - Rect.top)/3;
       } else {
           StartX = (Rect.right - Rect.left)/3;
           StartY = (Rect.bottom - Rect.top)/3;
       }
    }
  }

  LineLength = (Rect.bottom - Rect.top)/3 ;

  InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);

  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_VERTAXIS:

  ClearWindow (hWnd);

  GetClientRect (hWnd, (RECT FAR*) &Rect);

  /* Work out the ideal location */
```

```
    StartX = (Rect.right - Rect.left)/4 + Rect.left;

    StartY = (Rect.bottom - Rect.top)/4 + Rect.top;

    LineLength = (Rect.bottom - Rect.top)/2 ;

    VertAxis (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                    iHarmony, iAccuracy);

    return (DefWindowProc (hWnd, message, wParam, lParam));

  case IDM_HORIZAXIS:

    ClearWindow (hWnd);

    GetClientRect (hWnd, (RECT FAR*) &Rect);

    /* Work out the ideal location */

    StartX = (Rect.right - Rect.left)/4 + Rect.left;

    StartY = Rect.bottom - (Rect.bottom - Rect.top)/4;

    LineLength = (Rect.right - Rect.left)/2 ;

    HorizAxis (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                    iHarmony, iAccuracy);

    return (DefWindowProc (hWnd, message, wParam, lParam));

  case IDM_GRAPHAXES:

    ClearWindow (hWnd);

    GetClientRect (hWnd, (RECT FAR*) &Rect);

    /* Work out the ideal location */
```

```
  StartX = (Rect.right - Rect.left)/4 + Rect.left;

  StartY = (Rect.bottom - Rect.top)/4 + Rect.top;
  LineLength = (Rect.bottom - Rect.top)/2 ;

  VertAxis (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);

  StartX = EndX - 5;
  StartY = EndY - 5;
  LineLength *= 3;
  LineLength /= 2;
  HorizAxis (hWnd, iShakiness, iPeriod, iDirection, iThickness, MAX_HARMONY,
              iAccuracy);

return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_SQUARE:

  ClearWindow (hWnd);
  GetClientRect (hWnd, (RECT FAR*) &Rect);  /* Get size of window */
  StartX = (Rect.right - Rect.left)/3;
  StartY = (Rect.bottom - Rect.top)/3;

  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 0;
  InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;

  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 270;
  InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;
```

```
  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 180;
 InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;

  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 90;
 InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;

  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_TRIANGLE:

  ClearWindow (hWnd);
  GetClientRect (hWnd, (RECT FAR*) &Rect);  /* Get size of window */
  StartX = (Rect.right - Rect.left)/3;
  StartY = (Rect.bottom - Rect.top)/3;

  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 300;
 InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;

  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 180;
 InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);
  StartX = EndX;
  StartY = EndY;
```

```
  LineLength = (Rect.bottom - Rect.top)/3;
  iDirection = 60;
 InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
                  iHarmony, iAccuracy);

  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_VERTICALS:
  ClearWindow (hWnd);

  GetClientRect (hWnd, (RECT FAR*) &Rect);  /* Get size of window */
  DiffX = (Rect.right - Rect.left)/NUM_VERTICAL_LINES;
  StartY = (Rect.bottom - Rect.top)/20;

  LineLength = (Rect.bottom - Rect.top) - StartY*2;
  iDirection = 270;

  for (i = 1; i < NUM_VERTICAL_LINES; i++) {
    StartX = DiffX * i;
    InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
            iHarmony, iAccuracy);
  }

  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_HORIZONTALS:

  ClearWindow (hWnd);

  GetClientRect (hWnd, (RECT FAR*) &Rect);  /* Get size of window */
  StartX = (Rect.right - Rect.left)/20;
  DiffY = (Rect.bottom - Rect.top)/NUM_HORIZONTAL_LINES;

  LineLength = (Rect.right - Rect.left) - StartX*2;
  iDirection = 0;

  for (i = 1; i < NUM_HORIZONTAL_LINES; i++) {
    StartY = DiffY * i;
```

```
      InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
              iHarmony, iAccuracy);
    }

    return (DefWindowProc (hWnd, message, wParam, lParam));

  case IDM_GRID:

    ClearWindow (hWnd);

    GetClientRect (hWnd, (RECT FAR*) &Rect);  /* Get size of window */

    DiffX = (Rect.right - Rect.left)/NUM_VERTICAL_LINES;
    StartY = (Rect.bottom - Rect.top)/20;

    LineLength = (Rect.bottom - Rect.top) - StartY*2;
    iDirection = 270;

    for (i = 1; i < NUM_VERTICAL_LINES; i++) {
      StartX = DiffX * i;
      InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
              iHarmony, iAccuracy);
    }

    StartX = (Rect.right - Rect.left)/20;
    DiffY = (Rect.bottom - Rect.top)/NUM_HORIZONTAL_LINES;

    LineLength = (Rect.right - Rect.left) - StartX*2;
    iDirection = 0;

    for (i = 1; i < NUM_HORIZONTAL_LINES; i++) {
      StartY = DiffY * i;
      InformalLine (hWnd, iShakiness, iPeriod, iDirection, iThickness,
              iHarmony, iAccuracy);
    }

    return (DefWindowProc (hWnd, message, wParam, lParam));

  case IDM_CLEAR:
```

```
  ClearWindow (hWnd);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_RESET:
  ResetDefaults ();
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_EXIT:
  SendMessage (hWnd, WM_CLOSE, 0, 0L);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_SHAKINESS:
  lpProc = MakeProcInstance ((FARPROC) GetShakiness, hinst);
  DialogBox (hinst, "ShakinessBox", hWnd, (FARPROC) lpProc);
  FreeProcInstance (lpProc);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_THICKNESS:
  lpProc = MakeProcInstance ((FARPROC) GetThickness, hinst);
  DialogBox (hinst, "ThicknessBox", hWnd, (FARPROC) lpProc);
  FreeProcInstance (lpProc);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_PERIOD:
  lpProc = MakeProcInstance ((FARPROC) GetPeriod, hinst);
  DialogBox (hinst, "PeriodBox", hWnd, (FARPROC) lpProc);
  FreeProcInstance (lpProc);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_DIRECTION:
  lpProc = MakeProcInstance ((FARPROC) GetDirection, hinst);
  DialogBox (hinst, "DirectionBox", hWnd, (FARPROC) lpProc);
  FreeProcInstance (lpProc);
  return (DefWindowProc (hWnd, message, wParam, lParam));

case IDM_HARMONY:
  lpProc = MakeProcInstance ((FARPROC) GetHarmony, hinst);
  DialogBox (hinst, "HarmonyBox", hWnd, (FARPROC) lpProc);
```

```
            FreeProcInstance (lpProc);
            return (DefWindowProc (hWnd, message, wParam, lParam));

          case IDM_ACCURACY:
            lpProc = MakeProcInstance ((FARPROC) GetAccuracy, hinst);
            DialogBox (hinst, "AccuracyBox", hWnd, (FARPROC) lpProc);
            FreeProcInstance (lpProc);
            return (DefWindowProc (hWnd, message, wParam, lParam));

          default:
            return (DefWindowProc (hWnd, message, wParam, lParam));
          }

      case WM_CREATE:
        hAccTable = LoadAccelerators (hinst, "i2obMenu");
        break;

      case WM_DESTROY:
        PostQuitMessage (0);
        break;

      default:
        return (DefWindowProc (hWnd, message, wParam, lParam));
    }
    return NULL;
}



/*
       Routine to draw a vertical line
*/

void FAR PASCAL VerticalLine (HWND hWnd, int Shakiness, int Period, int
Direction, int Thickness, int Harmony, int Accuracy)
{
    int s, t, n, p;
    HDC hDC;
    char str [4];
```

```
  hDC = GetDC (hWnd);

  DisplayDims ();


  DoAccuracy (Accuracy);

  DoHarmonyX (Harmony);
  /* Draw the line */

  s = random (Shakiness) - 1;

  for (n = 0; n < LineLength; n++) {

    for (p = 0; p <= Period; p++) {

      for (t = 0; t < Thickness; t++)
       SetPixel (hDC, StartX+s+t, StartY+n, SET_PIXEL);
      n++;
    }
    n--;
    s = random (Shakiness) - 1;
  }

  EndX = StartX+s+t-1;
  EndY = StartY+n-1;

  ReleaseDC (hWnd, hDC);

  return;
}



/*
  Main routine to draw an informal line
*/

void FAR PASCAL InformalLine (HWND hWnd, int Shakiness, int Period, int
Direction, int Thickness, int Harmony, int Accuracy)
```

```
{
  int s, t, n, p;
  HDC hDC;
  char str [4];
  double Angle, xt, yt;
  int x, y;

  hDC = GetDC (hWnd);

  x = StartX;
  y = StartY;

  Angle = Direction;

  Angle = fmod (Angle, 90);

  if ((Direction == 90) || (Direction == 180) || (Direction == 270))
    Angle = 90;

  Angle = DegToRad (Angle);

  DisplayDims ();

/*
  DoAccuracy (Accuracy);

  DoHarmonyX (Harmony);
*/

  s = random (Shakiness) - 1;

  if (Direction < 45) {

    /* 0 < angle < 45 */

    LineLength = LineLength * cos (Angle);
    xt = 0;
```

```
  for (n = 0; n < LineLength; n++) {

    for (p = 0; p <= Period; p++) {
     for (t = 0; t < Thickness; t++)
       SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

     x++;
     xt++;

     if (xt * (tan (Angle)) >= 1) {          /* rounding problems */
       y--;
       xt = 0;
     }
     n++;
    }
    n--;
    s = random (Shakiness) - 1;
  }

} else {

  if (Direction <= 90) {

    /* 45 < angle < 90 */

    LineLength = LineLength * sin (Angle);
    yt = 0;

    for (n = 0; n < LineLength; n++) {

     for (p = 0; p <= Period; p++) {

       for (t = 0; t < Thickness; t++)
         SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

       y--;
       yt++;
```

```
      if (yt / (tan (Angle)) >= 1) {
        x++;
        yt = 0;
      }

      n++;
    }

    n--;
    s = random (Shakiness) - 1;
   }

} else {

   if (Direction < 135 ) {

    /* 90 < angle < 135 */

    LineLength = LineLength * cos (Angle);
    yt = 0;

    for (n = 0; n < LineLength; n++) {

      for (p = 0; p <= Period; p++) {

        for (t = 0; t < Thickness; t++)
          SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

        y--;
        yt++;

        if (yt * (tan (Angle)) >= 1) {
          x--;
          yt = 0;
        }
        n++;
      }
      n--;
```

```
  s = random (Shakiness) - 1;
 }

} else {

 if (Direction <= 180 ) {

   /* 135 < angle < 180 */

   LineLength = LineLength * sin (Angle);
   xt = 0;

   for (n = 0; n < LineLength; n++) {

     for (p = 0; p <= Period; p++) {

       for (t = 0; t < Thickness; t++)
        SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

       x--;
       xt++;

       if (xt / (tan (Angle)) >= 1) {
        y--;
        xt = 0;
       }
       n++;
     }
     n--;
     s = random (Shakiness) - 1;
   }

 } else {
   if (Direction < 225 ) {
     /* 180 < angle < 225 */
     LineLength = LineLength * cos (Angle);
     xt = 0;
     for (n = 0; n < LineLength; n++) {
```

```
    for (p = 0; p <= Period; p++) {
     for (t = 0; t < Thickness; t++)
       SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);
     x--;
     xt++;
     if (xt * (tan (Angle)) >= 1) {
       y++;
       xt = 0;
     }
     n++;
    }
    n--;
    s = random (Shakiness) - 1;
  }

} else {

  if (Direction <= 270 ) {

    /* 225 < angle < 270 */

    LineLength = LineLength * sin (Angle);
    yt = 0;

    for (n = 0; n < LineLength; n++) {

     for (p = 0; p <= Period; p++) {

       for (t = 0; t < Thickness; t++)
         SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

       y++;
       yt++;

       if (yt / (tan (Angle)) >= 1) {
         x--;
         yt = 0;
       }
```

```
      n++;
     }
     n--;
     s = random (Shakiness) - 1;
    }


  } else {

    if (Direction < 315 ) {

     /* 270 < angle < 315 */

     LineLength = LineLength * cos (Angle);
     yt = 0;

     for (n = 0; n < LineLength; n++) {

       for (p = 0; p <= Period; p++) {

         for (t = 0; t < Thickness; t++)
           SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

         y++;
         yt++;

         if (yt * (tan (Angle)) >= 1) {
           x++;
           yt = 0;
         }
         n++;
       }
       n--;
       s = random (Shakiness) - 1;
     }

    } else {
```

```
                /* 315 < angle < 360 */


                LineLength = LineLength * sin (Angle);
                xt = 0;

                for (n = 0; n < LineLength; n++) {

                  for (p = 0; p <= Period; p++) {

                    for (t = 0; t < Thickness; t++)
                     SetPixel (hDC, x+s+t, y+s+t, SET_PIXEL);

                    x++;
                    xt++;

                    if (xt / (tan (Angle)) >= 1) {
                       y++;
                       xt = 0;
                      }

                    n++;
                    }
                    n--;
                    s = random (Shakiness) - 1;
                  }
                }
              }
            }
          }
        }
      }
    }

EndX = x;
EndY = y;
```

```
  ReleaseDC (hWnd, hDC);

  return;
}

void FAR PASCAL VertAxis (HWND hWnd, int Shakiness, int Period, int Direction,
int Thickness, int Harmony, int Accuracy)
{
  int s, t, n, p;
  HDC hDC;

  randomize ();

  DisplayDims ();

  DoAccuracy (Accuracy);

  DoHarmonyX (Harmony);

  hDC = GetDC (hWnd);

  s = random (Shakiness) - 1;

  for (n = 0; n < LineLength; n++) {

    for (p = 0; p <= Period; p++) {

      for (t = 0; t < Thickness; t++)
       SetPixel (hDC, StartX+s+t, StartY+n, SET_PIXEL);
      n++;
    }
    n--;
    s = random (Shakiness) - 1;
  }

  ReleaseDC (hWnd, hDC);

  EndX = StartX+s+t-1;
  EndY = StartY+n-1;
```

```
  return;
}

void FAR PASCAL HorizAxis (HWND hWnd, int Shakiness, int Period, int Direction,
int Thickness, int Harmony, int Accuracy)
{
  int s, t, n, p;
  HDC hDC;

  randomize ();

  DisplayDims ();

  DoAccuracy (Accuracy);

  DoHarmonyY (Harmony);

  hDC = GetDC (hWnd);

  s = random (Shakiness) - 1;

  for (n = 0; n < LineLength; n++) {

    for (p = 0; p <= Period; p++) {

      for (t = 0; t < Thickness; t++)
       SetPixel (hDC, StartX+n, StartY+s+t, SET_PIXEL);
      n++;
    }
    n--;
    s = random (Shakiness) - 1;
  }

  ReleaseDC (hWnd, hDC);

  EndX = StartX+n-1;
  EndY = StartY+s+t-1;
```

```
  return;
}

void FAR PASCAL DoAccuracy (int Accuracy)
{
  int n;

  Accuracy = 10 - Accuracy;       /* now 0 - 10 */

  n = rand () % 2;

  if (n == 0) {
    StartX = StartX + Accuracy;
  } else {
    StartX = StartX - Accuracy;
  }

  n = rand () % 2;
  if (n == 0) {
    StartY = StartY + Accuracy;
  } else {
    StartY = StartY - Accuracy;
  }


  n = rand () % 2;
  if (n == 0) {
    LineLength = LineLength + Accuracy;
  } else {
    LineLength = LineLength - Accuracy;
  }

  return;
}

void FAR PASCAL DoHarmonyX (Harmony)
{
  int n, nn;
```

```
  float fHarmony, f;

  /* Now introduce disharmony */

  fHarmony = Harmony;
  fHarmony = 1 - (fHarmony / 10);

/*  randomize (); */

  n = rand () % StartX;

  f = n;
  f = fHarmony * f;

  nn = f;

  if (n < StartX/2) {
    StartX = StartX - nn;
  } else {
    StartX = StartX + nn;
  }
  return;
}

void FAR PASCAL DoHarmonyY (Harmony)
{
  int n, nn;
  float fHarmony, f;

  /* Now introduce disharmony */

  fHarmony = Harmony;
  fHarmony = 1 - (fHarmony / 10);

/*  randomize (); */

  n = rand () % StartY;
```

```
  f = n;
  f = fHarmony * f;

  nn = f;

  if (n < StartY/2) {
    StartY = StartY - nn;
  } else {
    StartY = StartY + nn;
  }
  return;
}
```

i-Fax Source Code Listing (C++)

```
TPoint IFaxWindow::DrawLine(int s, int p, int d, int t, int h, int a, int l,
TPoint sp)
{
       TPoint point;
       xPorter *xp;

// create a new ILine object:
       ILine il(s, p, d, t, h, a, l, sp);

       xp->xPortWriteIlProperties(s, p, d, t, h, a, l, sp);

       point = il.GetPoint();

       if (!DragDC) {
              SetCapture();
              DragDC = new TClientDC(*this);
              DragDC->MoveTo(point);

              while (point != TPoint(0xFFFF, 0xFFFF)) {
                point = il.GetPoint();
                if (point != TPoint(0xFFFF, 0xFFFF))
                       DragDC->LineTo(point);
              };
              ReleaseCapture();
```

```
            delete DragDC;
            DragDC = 0;
        }

      point = il.GetEndPoint();

      return (point);
}

// CmProcessBuffer
//
// Process the contents of the TPointsIterator buffer, trying to
// determine what sort of informal object it may be

void IFaxWindow::CmProcessBuffer(void)
{
  TPointsIterator i(*Line);
  TPoint endp;
  int xdiff, ydiff;
  int maxx, maxy, mx, my;
  int Direction;
  char s[80];
  int dirn;
  xPorter *xp;
  int per;

  Direction = DONT_KNOW;

  per = 0;                        // period;

  maxx = 0;
  maxy = 0;

  TPoint startp = i++;

  while (i) {

      TPoint p = i++;
```

```
        mx = abs(p.x - startp.x);  // go along the points to work
                                   // out the variance
        if (maxx < mx)
              maxx = mx;

        if ((p.x == startp.x) && (p.x != endp.x))
              per++;

        my = abs(p.y - startp.y);

        if (maxy < my)
              maxy = my;

        if ((p.y == startp.y) && (p.y != endp.y))
              per++;

        endp = p;                  // save the last point
  }

// Don't want a zero period:

  if (!per)
        per++;

  xdiff = endp.x - startp.x;
  ydiff = endp.y - startp.y;



  if ((ydiff < MAX_HORIZ_YDIFF) && (maxy < MAXY))
        Direction = HORIZONTAL;

  if ((xdiff < MAX_VERT_XDIFF) && (maxx < MAXX))
        Direction = VERTICAL;

  switch (Direction) {

        case (HORIZONTAL):
```

```
            if (endp.x > startp.x) {
              dirn = 90;
            } else {
              dirn = 270;
            }

// Shakiness, Period, Direction,Thickness, Harmony, Accuracy,
// Length, Start
            xp->xPortWriteIlProperties(maxy, per, dirn, 1, 0, 0, maxx, startp);

            break;

      case (VERTICAL):

            if (endp.y > startp.y) {
              dirn = 180;
            } else {
              dirn = 0;
            }

            xp->xPortWriteIlProperties(maxx, per, dirn, 1, 0, 0, maxy,
                  startp);
            break;
      default:
            break;
  }

void IFaxWindow::CmDisplayBuffer(void)
{
  bool first = true;
  TPointsIterator i(*Line);

  if (!DragDC) {
            SetCapture();
            DragDC = new TClientDC(*this);

      while (i) {
```

```
            TPoint p = i++;

            if (!first)
              DragDC->LineTo(p);
            else {
              DragDC->MoveTo(p);
              first = false;
            }
        }
        ReleaseCapture();
        delete DragDC;
        DragDC = 0;
  }
  return;
}


// Constructor for ILine:
ILine::ILine(int s, int p, int d, int t, int h, int a, int l, TPoint sp)
{
       IShakiness = s;
       IPeriod = p;
       IDirection = d;
       IThickness = t;
       IHarmony = h;
       IAccuracy = a;
       ILength = l;
       StartPoint = sp;
       EndPoint = sp;
       NextFlag = 0;
       length = 0;
       rAngle = 0;
       PrototypePoint = 0;
       period = IPeriod;
}

// ILine:: GetEndPoint; accessor for EndPoint
```

```
TPoint ILine::GetEndPoint()
{
  return (EndPoint);
}

int GetSFactor(int sr)
{
  int s;

  sr = random(sr);

  s = random(2);                          // decide whether to + or -

  if (!s)
        sr = -sr;

  return (sr);
}



// ILine::GetPoint
// Angles are measured CLOCKWISE from the vertical

TPoint ILine::GetPoint()
{
  int sr;

  // Get the shakiness factor:

  sr = GetSFactor(IShakiness);

  if (IThickness == 0)
        return (TPoint(0xFFFF, 0xFFFF));

  if (!NextFlag) {
        Point = StartPoint;
        PrototypePoint = Point;
        NextFlag = 1;
```

```
        return (StartPoint);
  }

  if (length >= ILength) {
        EndPoint = Point;
        return (TPoint(0xFFFF, 0xFFFF));
  }

switch(IDirection) {

        case (0):
              if (period--) {            // wait for period to come down
                                         //to zero before
                Point-= TPoint(0, 1);    // we decide whether to do a jiggle
                PrototypePoint-= TPoint(0, 1);
                break;
              } else {
                Point-= TPoint(sr, 1);
                PrototypePoint-= TPoint(0, 1);
                period = IPeriod;
                break;
              }

        case (45):
              if (period--) {
                Point+= TPoint(1, 0);
                sr = GetSFactor(IShakiness);
                Point-= TPoint(0, 1);
                PrototypePoint+= TPoint(1, 0);
                PrototypePoint-= TPoint(0, 1);
                break;
              } else {
                Point+= TPoint(1, sr);
                sr = GetSFactor(IShakiness);
                Point-= TPoint(sr, 1);
                PrototypePoint+= TPoint(1, 0);
                PrototypePoint-= TPoint(0, 1);
                period = IPeriod;
```

```
        break;
      }

case (90):
      if (period--) {
        Point+= TPoint(1, 0);
        PrototypePoint+= TPoint(1, 0);
        break;
      } else {
        Point+= TPoint(1, sr);
        PrototypePoint+= TPoint(1, 0);
        period = IPeriod;
        break;
      }

case (135):
      if (period--) {
        Point+= TPoint(1, 0);
        sr = GetSFactor(IShakiness);
        Point+= TPoint(0, 1);
        PrototypePoint+= TPoint(1, 0);
        PrototypePoint+= TPoint(0, 1);
        break;
      } else {
        Point+= TPoint(1, sr);
        sr = GetSFactor(IShakiness);
        Point+= TPoint(sr, 1);
        PrototypePoint+= TPoint(1, 0);
        PrototypePoint+= TPoint(0, 1);
        period = IPeriod;
        break;
      }

case (180):
      if (period--) {
        Point+= TPoint(0, 1);
        PrototypePoint+= TPoint(0, 1);
        break;
```

```
      } else {
        Point+= TPoint(sr, 1);
        PrototypePoint+= TPoint(0, 1);
        period = IPeriod;
        break;
      }

case (225):
      if (period--) {
        Point-= TPoint(1, 0);
        sr = GetSFactor(IShakiness);
        Point+= TPoint(0, 1);
        PrototypePoint-= TPoint(1, 0);
        PrototypePoint+= TPoint(0, 1);
        break;
      } else {
        Point-= TPoint(1, sr);
        sr = GetSFactor(IShakiness);
        Point+= TPoint(sr, 1);
        PrototypePoint-= TPoint(1, 0);
        PrototypePoint+= TPoint(0, 1);
        period = IPeriod;
        break;
      }

case (270):
      if (period--) {
        Point-= TPoint(1, 0);
        PrototypePoint-= TPoint(1, 0);
        break;
      } else {
        Point-= TPoint(1, sr);
        PrototypePoint-= TPoint(1, 0);
        period = IPeriod;
        break;
      }

case (315):
```

```
            if (period--) {
              Point-= TPoint(1, 0);
              sr = GetSFactor(IShakiness);
              Point-= TPoint(0, 1);
              PrototypePoint-= TPoint(1, 0);
              PrototypePoint-= TPoint(0, 1);
              break;
            } else {
              Point-= TPoint(1, sr);
              sr = GetSFactor(IShakiness);
              Point-= TPoint(sr, 1);
              PrototypePoint-= TPoint(1, 0);
              PrototypePoint-= TPoint(0, 1);
              period = IPeriod;
              break;
            }

        default:
              break;
  }

  length++;

  return (Point);
}

TPoint IFaxWindow::DrawSquare(TPoint StartPoint, int length)
{
  TPoint point;

// DrawLine returns TPoint end point, the last point drawn

// Shakiness, Period, Direction,Thickness, Harmony, Accuracy, Length, Start
  point = DrawLine(2, 4, 90,  3, 0, 0, length, StartPoint);
  point = DrawLine(2, 4, 180, 3, 0, 0, length, point);
  point = DrawLine(2, 4, 270, 3, 0, 0, length, point);
  point = DrawLine(2, 4, 0,   3, 0, 0, length, point);
  return (point);
```

```
}

void IFaxWindow::CmSquare(void)
{
  DrawSquare(STARTSQUARE, SIZESQUARE);
  return;
}

void IFaxWindow::CmStar()
{
  DrawLine(2, 4, 0,   3, 0, 0, 90, STARTSTAR);
  DrawLine(2, 4, 45,  3, 0, 0, 60, STARTSTAR);
  DrawLine(2, 4, 90,  3, 0, 0, 90, STARTSTAR);
  DrawLine(2, 4, 135, 3, 0, 0, 60, STARTSTAR);
  DrawLine(2, 4, 180, 3, 0, 0, 90, STARTSTAR);
  DrawLine(2, 4, 225, 3, 0, 0, 60, STARTSTAR);
  DrawLine(2, 4, 270, 3, 0, 0, 90, STARTSTAR);
  DrawLine(2, 4, 315, 3, 0, 0, 60, STARTSTAR);
  return;
}

void IFaxWindow::CmHouse()
{
  TPoint tp, ChimneyPlace, ChimneyOffset, WindowPosition;

// draw the main building square
  tp = DrawSquare(STARTHOUSE, SIZEHOUSE);

// then draw the roof
  tp = DrawLine(2, 4, 45,  3, 0, 0, 25, tp);
  tp = DrawLine(2, 4, 90,  3, 0, 0, SIZEHOUSE/2, tp);
  ChimneyPlace = tp;
  DrawLine(2, 4, 135, 3, 0, 0, 25, tp);

// and windows

  WindowPosition = TPoint(SIZEHOUSE/5, SIZEHOUSE/5);
  WindowPosition += STARTHOUSE;
```

```
  DrawSquare(WindowPosition, SIZEWINDOW);

  WindowPosition = TPoint((SIZEHOUSE - (2*SIZEHOUSE/5)),  SIZEHOUSE/5);
  WindowPosition += STARTHOUSE;
  DrawSquare(WindowPosition, SIZEWINDOW);

  WindowPosition = TPoint(SIZEHOUSE/5, (SIZEHOUSE - (2*SIZEHOUSE/5)));
  WindowPosition += STARTHOUSE;
  DrawSquare(WindowPosition, SIZEWINDOW);

  WindowPosition = TPoint((SIZEHOUSE - (2*SIZEHOUSE/5)), (SIZEHOUSE –
  (2*SIZEHOUSE/5)));
  WindowPosition += STARTHOUSE;
  DrawSquare(WindowPosition, SIZEWINDOW);

// and finally a chimney
  ChimneyOffset = TPoint(-SIZECHIMNEY*2, -SIZECHIMNEY);
  ChimneyPlace = ChimneyPlace + ChimneyOffset;

  DrawSquare(ChimneyPlace, SIZECHIMNEY);

  return;
}
```

Examiner Source Code Listing (Prolog)

```
main :-
      openlog('examiner.log'),
      write($EXAMINER Context Expert START$), nl,
      write($Copyright (c) 1996 Ian Cullimore. All rights reserved.$),
      nl,
      write($Revision:   1.00$), nl,
      consult($\\out\\ifax.pro$),
      examiner,
      closelog.

examiner :-
      write($Examining...$), nl,
      write($Found objects:$), nl,
      objecttype(Handle, Type),
      write($handle: $),
      write(Handle),
      write($; type: $),
      write(Type),
      direction(Handle, D),
      write($; Dirn: $),
      write(D),
      nl,
      fail.

examiner :-
      get_joins,
      fail.

examiner :-
      find_shapes,
      fail.

examiner :-
      write($EXAMINER Context Expert STOP$), nl.
```

```
% ------------------------------------------------------
%
% find_shapes

find_shapes :-
      find_four_sided_shape,
      fail.

find_shapes :-
      find_triangle,
      fail.

find_shapes :-
      find_rectangle,
      fail.

find_triangle :-
      joined(A, B),
      joined(B, C),
      joined(C, A),
      write($Found a triangle$), nl.

find_rectangle :-
      joined(A, B),
      right_angle(A, B),
      joined(B, C),
      right_angle(B, C),
      joined(C, D),
      right_angle(C, D),
      joined(D, A),
      right_angle(D, A),
      write($Found a rectangle from $),
      write(A), write($ to $), write(B), write($ to $),     write(C),
write($ to $), write(D),
      nl.

find_four_sided_shape :-
      joined(A, B),
      joined(B, C),
```

```
      joined(C, D),
      joined(D, A),
      write($Found a four-sided shape from $),
      write(A), write($ to $), write(B), write($ to $),     write(C),
write($ to $), write(D),
      nl.


% ------------------------------------------------------
%
% get_joins
%
% Search for joined lines


get_joins :-
      joined_to,
      fail.


get_joins :-
      joined(X, Y),
      write($Found a join from $), write(X), write($ to     $),
write(Y), nl,
      fail.


% ------------------------------------------------------
%
% joined_to


% direction 0
joined_to :-
      objecttype(Handle, line),
      direction(Handle, 0),
      length(Handle, Length),
      accuracy(Handle, Accuracy),
      AA is Accuracy*2,
      startpoint(Handle, Xcoord, Ycoord),
      XXcoord is Xcoord - Accuracy,
      YYcoord is Ycoord - Length - Accuracy,
      for(X, 0, AA, 1),
      for(Y, 0, AA, 1),
```

```
      XX is XXcoord + X,
      YY is YYcoord + Y,
      startpoint(Handle1, XX, YY),
      assert(joined(Handle, Handle1)),
      direction(Handle, D0),
      direction(Handle1, D1),
      (D1 - D0) =:= 90,
      assert(right_angle(Handle, Handle1)),
      fail.


% direction 90

joined_to :-
      objecttype(Handle, line),
      direction(Handle, 90),
      writelog($Found direction 90$), nllog,
      length(Handle, Length),
      accuracy(Handle, Accuracy),
      AA is Accuracy*2,
      writelog($length: $),
      writelog(Length), nllog,
      startpoint(Handle, Xcoord, Ycoord),
      writelog($StartPoint: $),
      writelog(Xcoord), writelog($; $), writelog(Ycoord),   nllog,
      writelog($EndPoint: $),
      XXcoord is Xcoord + Length - Accuracy,
      YYcoord is Ycoord - Accuracy,
      writelog(EndX), writelog($; $), writelog(Ycoord),     nllog,
      for(X, 0, AA, 1),
      for(Y, 0, AA, 1),
%     writelog($X:$), writelog(X), nllog,
%     writelog($Y:$), writelog(Y), nllog,
      XX is XXcoord + X,
      YY is YYcoord + Y,
%     writelog($XX:$), writelog(XX), nllog,
%     writelog($YY:$), writelog(YY), nllog,
      startpoint(Handle1, XX, YY),
```

```
%     writelog(XX), nllog,
%     writelog(YY), nllog,
      assert(joined(Handle, Handle1)),
      direction(Handle, D0),
      direction(Handle1, D1),
      (D1 - D0) =:= 90,
      assert(right_angle(Handle, Handle1)),
      fail.

% direction 180

joined_to :-
      objecttype(Handle, line),
      direction(Handle, 180),
      writelog($Found direction 180$), nllog,
      length(Handle, Length),
      writelog($length: $),
      writelog(Length), nllog,
      accuracy(Handle, Accuracy),
      AA is Accuracy*2,
      startpoint(Handle, Xcoord, Ycoord),
      writelog($StartPoint: $),
      writelog(Xcoord), writelog($; $), writelog(Ycoord),   nllog,
      writelog($EndPoint: $),
      XXcoord is Xcoord - Accuracy,
      YYcoord is Ycoord + Length - Accuracy,
      writelog(Xcoord), writelog($; $), writelog(EndY),     nllog,
      for(X, 0, AA, 1),
      for(Y, 0, AA, 1),
      XX is XXcoord + X,
      YY is YYcoord + Y,
      startpoint(Handle1, XX, YY),
      assert(joined(Handle, Handle1)),
      direction(Handle, D0),
      direction(Handle1, D1),
      (D1 - D0) =:= 90,
      assert(right_angle(Handle, Handle1)),
      fail.
```

```
% direction 270

joined_to :-
      objecttype(Handle, line),
      direction(Handle, 270),
      writelog($Found direction 270$), nllog,
      length(Handle, Length),
      writelog($length: $),
      writelog(Length), nllog,
      accuracy(Handle, Accuracy),
      AA is Accuracy*2,
      startpoint(Handle, Xcoord, Ycoord),
      writelog($StartPoint: $),
      writelog(Xcoord), writelog($; $), writelog(Ycoord),   nllog,
      writelog($EndPoint: $),
      XXcoord is Xcoord - Length - Accuracy,
      YYcoord is Ycoord - Accuracy,
      writelog(EndX), writelog($; $), writelog(Ycoord),     nllog,
      for(X, 0, AA, 1),
      for(Y, 0, AA, 1),
      XX is XXcoord + X,
      YY is YYcoord + Y,
      startpoint(Handle1, XX, YY),
      assert(joined(Handle, Handle1)),
      direction(Handle, D0),
      direction(Handle1, D1),
      (D1 - D0) =:= -270,
      assert(right_angle(Handle, Handle1)),
      fail.
```

Appendix B

Example Output From EXAMINER

```
EXAMINER Context Expert START
Copyright (c) 1996 Ian Cullimore. All rights reserved.
Revision:   1.00
objecttype(hBBBICOAA,line)
shakiness(hBBBICOAA,2)
period(hBBBICOAA,4)
direction(hBBBICOAA,90)
thickness(hBBBICOAA,3)
harmony(hBBBICOAA,0)
accuracy(hBBBICOAA,5)
length(hBBBICOAA,90)
startpoint(hBBBICOAA,256,256)
objecttype(hBBBICOAB,line)
shakiness(hBBBICOAB,2)
period(hBBBICOAB,4)
direction(hBBBICOAB,180)
thickness(hBBBICOAB,3)
harmony(hBBBICOAB,0)
accuracy(hBBBICOAB,5)
length(hBBBICOAB,90)
startpoint(hBBBICOAB,346,256)
objecttype(hBBBICOAC,line)
shakiness(hBBBICOAC,2)
period(hBBBICOAC,4)
direction(hBBBICOAC,270)
thickness(hBBBICOAC,3)
harmony(hBBBICOAC,0)
accuracy(hBBBICOAC,5)
length(hBBBICOAC,90)
startpoint(hBBBICOAC,349,346)
objecttype(hBBBICOAD,line)
shakiness(hBBBICOAD,2)
period(hBBBICOAD,4)
direction(hBBBICOAD,0)
thickness(hBBBICOAD,3)
harmony(hBBBICOAD,0)
accuracy(hBBBICOAD,5)
length(hBBBICOAD,90)
startpoint(hBBBICOAD,259,347)
'!EOF'
Examining...
Found objects:
handle: hBBBICOAA; type: line; Dirn: 90
handle: hBBBICOAB; type: line; Dirn: 180
handle: hBBBICOAC; type: line; Dirn: 270
handle: hBBBICOAD; type: line; Dirn: 0
$Found direction 90$
```

```
$length: $90
$StartPoint: $256$; $256
$EndPoint: $H422$; $256
$Found direction 180$
$length: $90
$StartPoint: $346$; $256
$EndPoint: $346$; $H432
$Found direction 270$
$length: $90
$StartPoint: $349$; $346
$EndPoint: $H422$; $346
Found a join from hBBBICOAC to hBBBICOAD
Found a join from hBBBICOAB to hBBBICOAC
Found a join from hBBBICOAA to hBBBICOAB
Found a join from hBBBICOAD to hBBBICOAA
Found a four-sided shape from hBBBICOAC to hBBBICOAD to hBBBICOAA to
hBBBICOAB
Found a four-sided shape from hBBBICOAB to hBBBICOAC to hBBBICOAD to
hBBBICOAA
Found a four-sided shape from hBBBICOAA to hBBBICOAB to hBBBICOAC to
hBBBICOAD
Found a four-sided shape from hBBBICOAD to hBBBICOAA to hBBBICOAB to
hBBBICOAC
Found a rectangle from hBBBICOAC to hBBBICOAD to hBBBICOAA to
hBBBICOAB
Found a rectangle from hBBBICOAB to hBBBICOAC to hBBBICOAD to
hBBBICOAA
Found a rectangle from hBBBICOAA to hBBBICOAB to hBBBICOAC to
hBBBICOAD
Found a rectangle from hBBBICOAD to hBBBICOAA to hBBBICOAB to
hBBBICOAC
Found a square from hBBBICOAC to hBBBICOAD to hBBBICOAA to hBBBICOAB
Found a square from hBBBICOAB to hBBBICOAC to hBBBICOAD to hBBBICOAA
Found a square from hBBBICOAA to hBBBICOAB to hBBBICOAC to hBBBICOAD
Found a square from hBBBICOAD to hBBBICOAA to hBBBICOAB to hBBBICOAC
EXAMINER Context Expert STOP
```

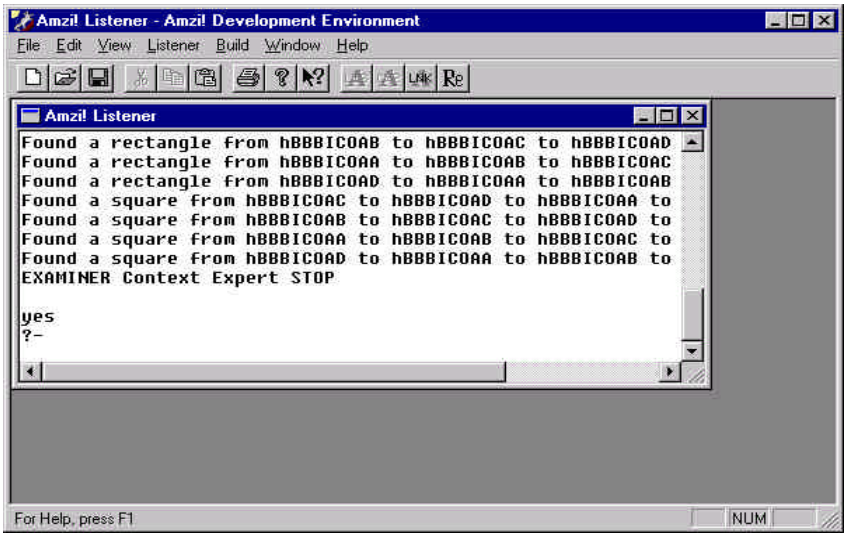

Figure 46: EXAMINER "discovering" a square

Appendix C

Object Definition for a Rough Straight Line in BNF Style

<object> ::= <object header> (<handle>, <parameter1>, <parameter2>,...,<parameterN>)

<object type> ::= “objecttype” | “shakiness” | “period” | “direction” | “thickness” | “harmony” | “accuracy” | “length” | “startpoint”

<handle> ::= <string> (unique)

<string> ::= <char> | <char><string>

<char> ::= <ASCII character>

if <object type> == “objecttype” -> N=1 && parameter1 == <objecttype>

<objecttype> == “line”

if <object type> == “shakiness” -> N=1 && parameter1 == ‘degree of shakiness’

if <object type> == “period” -> N=1 && parameter1 == ‘sine wave period of the line’

if <object type> == “direction” -> N=1 && parameter1 == ‘direction in degrees clockwise from the vertical’

if <object type> == “thickness” -> N=1 && parameter1 == ‘average pixel width of the line’

if <object type> == “harmony” -> N=1 && parameter1 == ‘harmonious placement of the object in the image field on a scale of 1 to 10’

if <object type> == “accuracy” -> N=1 && parameter1 == ‘precision of placement of the line’

if <object type> == “length” -> N=1 && parameter1 == ‘pixel length of the line’

if <object type> == “startpoint” -> N=2 && parameter1 == ‘pixel startpoint position measured from the top left hand corner’ && parameter2 == ‘pixel endpoint position measured from the top left hand corner’