A Method of Access to Computer Aided Software Engineering (CASE) Tools for Blind Software Engineers

Paul Blenkhorn and David Gareth Evans

Technology for Disabled People Unit, Systems Engineering Group,
Department of Computation, UMIST, Manchester, M60 1QD, England.

Abstract. This paper proposes a technique to allow blind software engineers to access the information held in Computer Aided Software Engineering (CASE) tools. Such tools support systems analysis and design methods that typically encode information as hierarchically structured two-dimensional graphs. The paper discusses the problems that blind engineers face in accessing this type of information by considering the structure of system models built in one Software Engineering notation, namely Hatley-Pirbhai Real Time Structured Analysis. It introduces a long established, but not widely used, notation, N^2 charts, which provide an equivalent tabular encoding for many software engineering notations. This style of presentation is used, together with talking touch tablet, to provide an interactive means for blind software engineers to access full system models.

1 Introduction

Many blind people work in the computer industry; the programming of computers is very largely text based and there are many well established methods for blind people to interact with text-based computer applications. However, there is very much more to developing a piece of software than simply writing the computer program. Software programs can be extremely complex and they are often used in safety-critical applications, for example in the control of aeroplanes, where failure in the software part of the system can have catastrophic effects. To control the complexity of the software program and to fully analyse its function, software-based systems are subjected to rigorous development methods. Such methods are termed Software Engineering methods.

When complex software systems were first developed, development methods consisted of a set of stages that resulted in a set of largely text-based documents. However, the problem with such methods is that natural language is a rather imprecise way of specifying definite concepts and that for complex systems these documents grow to considerable size, thus maintaining internal consistency in such documents is a formidable task. In the 1970s Software Engineering methods developed, initially for complex information systems, which specified the requirements and design of the system in, chiefly, graphical forms. The most significant method to be developed (in terms its of usage within the software industry) is Structured Analysis. This is used to capture system requirements and

carry out high level system design. Structured Analysis methods are widely used in a number forms, e.g. Yourdon [9], Hatley-Pirbhai [3] and Ward-Mellor [8]. These methods are broadly similar, and use notations to represent the developing system, which are almost identical. Over the past few years a set of similar methods, the Object-Oriented Software Engineering methods, have been developed (e.g. [2, 4, 7]); these share common goals with the Structured Analysis methods and the graphical notations to capture a system's requirements and to specify its design have strong similarities.

These methods are supported by a variety of computer-based tools (Computer Aided Software Engineering (CASE) Tools) supplied by a wide variety of vendors. These tools support the capture of the graphical models and carry out consistency checks to ensure that model is consistent with the set of rules specified by the development method.

The adoption of Software Engineering methods and the use of CASE tools presents a difficult problem for blind software engineers. The system analysis and design information is encoded in a two-dimensional graphical form making it extremely difficult for blind engineers to use standard methods of computer interaction. Moreover, as the next section describes, the information is presented in a hierarchical structure, the traversal of which is controlled by mouse-based 'point and click' techniques. In this paper we propose a method by which blind software engineers may access the information held in the graphical notations used in software engineering and show how this information is encoded. The proposed technique is currently under evaluation by the Technology for Disabled People Unit.

2 Software Engineering Notations

Initially we have chosen to restrict our work to the analysis phase of one of the Structured Analysis methods. The method chosen is the widely used Hatley-Pirbhai Real Time Structured Analysis method [3]. This method is used to analyse the requirements of the software and to carry out the high level design of the system. This particular method is chosen as it is popular and is supported by a number of CASE tools. The notation it uses is very similar to that of the other Structured Analysis methods and the technique presented in this paper is appropriate to the others. It is thought that object-oriented methods may also be handled by this technique, however they are somewhat different in detail to those described below.

The Hatley-Pirbhai Real Time Structured Analysis method is supported by a graphical notation, known as data flow diagrams (DFDs), which are used to describe the behaviour and the design of the system under development. A DFD presents a view of part of the system by dividing it into a number of processes that communicate with one another through data flows. A typical data flow diagram is shown in Fig. 1.

The circles (or "bubbles") on this diagram are processes; these take input data, process it and create output data. Data is routed from process to process through the directed arcs (the lines with arrows); these are named to describe the data that flows between the processes. The vertical bar is used to indicate a control specification associated with the DFD. Control information is routed into the control specification

through a set of directed arcs, which are represented by broken lines. These carry control information; i.e. they transmit, to other processes, the current state of the processes in the system. Control specifications are used to control the system by generating new control flows and by enabling and disabling the processes on the DFD, i.e. turning them off and on. Data stores are represented by parallel lines and have a name associated with them. These store persistent data in the system and are written to, and read from, by processes through unnamed data flows. Flows (both control and data) that do not have both ends connected to processes are connected at the immediately higher level in the hierarchy of DFDs.

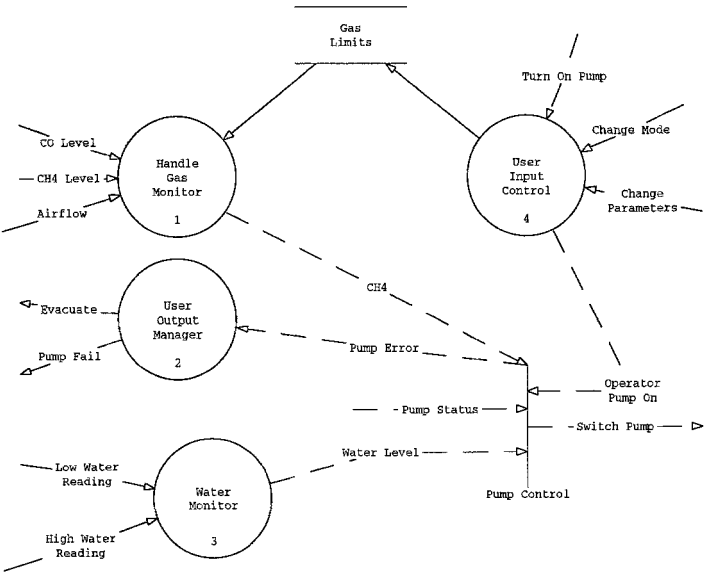

Fig. 1. A Typical Hatley-Pirbhai Data Flow Diagram

DFDs are connected into a hierarchy to form a complete model that represents the behaviour and design of the system. The hierarchical connections are formed by process refinement. Each process represented in a DFD at one level of the hierarchy can have its behaviour described by another DFD in the next layer of the hierarchy. At the top of this hierarchy there is a DFD with single process on it, which is connected by data flows to externals. These are elements at the boundary of the system, which provide input stimuli and accept output responses. This is called the context diagram because it sets the context for the system under development by describing the interface between it and the outside world. The decomposition of a system from context diagram is shown in Fig. 2. Processes are decomposed in this way until they are identified as being primitive, i.e. they are deemed simple enough not to warrant further decomposition. Their behaviour is described by a text specification, called a process specification, written in a language similar to a standard programming language. The control specifications are not decomposed in

the same way as the processes but have their behaviour described in a table or as a state transition diagram. Control specifications can be handled by the techniques proposed in this paper, however, due to space limitations these are not discussed further here. All data and control flows are described by text entries in a list called the Data Dictionary.

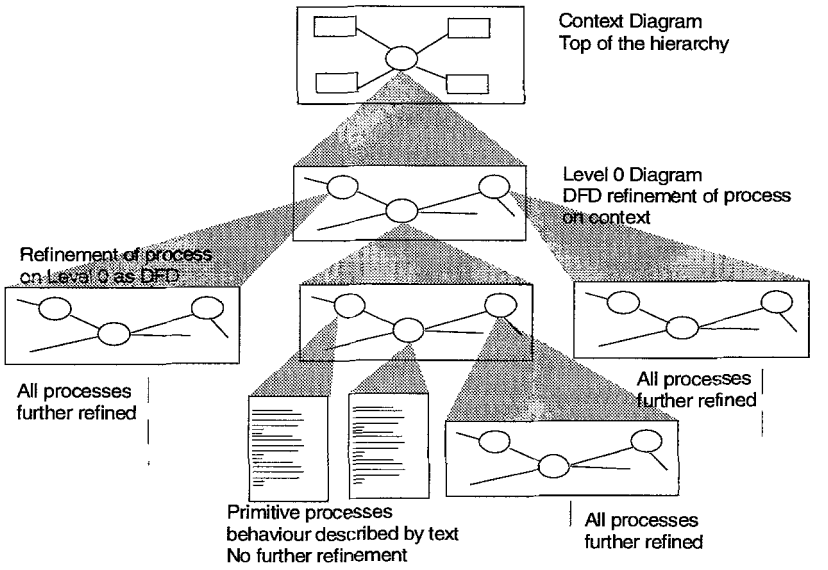

Fig. 2. The Hierarchical Structure of a Structured Analysis Model

When using a CASE tool, a sighted software engineer accesses the model of the system by moving up and down through the hierarchy. This is generally supported by the CASE tool in an interactive way, typically pointing to the process using a mouse pointer and clicking the mouse button will show the internal decomposition of the process, either as a DFD, or if primitive as a text specification. Data dictionary information is also accessed through this point and click mechanism.

Allowing blind software engineers to interact with captured Structured Analysis models requires: that the text be converted into a suitable media (here we use speech); and that the DFDs are redrawn, so that the notation *elements*[1] (i.e. processes, stores and control specifications) and *flows*[2] can be easily located and their associated text delivered. The notation used in the method proposed here is based upon N^2 charts, which are described below.

[1] In this paper the term *elements* is used to refer, collectively, to processes, stores and control specifications.

[2] *Flows* is used here to refer collectively to data flows, control flows and the connections to data stores.

3 N^2 Charts

There is an equivalent tabular form of any DFD; this is called an N^2 chart [5, 6]. These are not widely used in the software engineering industry, which generally favours more graphically based notations. The N^2 chart is a matrix with the *element* names running down the major diagonal. *Flows* between two *elements* are shown in the rectangles where the row and column of the connected *elements* intersect. Entries in the same row as an *element* are the outgoing flows from that *element*; entries in the same column are incoming data flows into the *element*. *Flows* that are connected at a higher level are placed on the N^2 chart to connect with a special *element* entry called the 'Upper Level'. The N^2 chart equivalent of Fig. 1 is presented in Fig. 3.

Upper Level	CO Level CH4 Level Airflow		Low Water Reading High Water Reading	Turn On Pump Change Mode Change Param-eters	Pump Status	
	Handle Gas Monitor				CH4	
Evacuate Pump Fail		User Input Manager				
			Water Monitor		Water Level	
				User Input Control	Operator Pump On	*Writes to* Gas Limits
Switch Pump		Pump Error			Pump Control	
	Reads from Gas Limits					Gas Limits

Fig. 3. N^2 chart equivalent of Fig. 1.

N^2 charts form the basis of the proposed method of access. Using a generic form of these charts and a talking touch window an evaluation system has been developed that allows a blind engineer to navigate through the full hierarchy of a Structured Analysis model.

4 The Evaluation System

The evaluation system is shown in Fig. 4. It consists of a UNIX workstation, which runs the Teamwork CASE tool and some custom software. A Touch Window and a speech synthesiser are connected to the workstation by serial lines. The user interacts with the system and interrogates the model by using the Touch Window. Most user actions cause the custom software to interrogate the database of the CASE tool by using a set of procedures provided in the Teamwork/ACCESS package [11]. In response to a user action, speech is generated that describes the selected portion of the model.

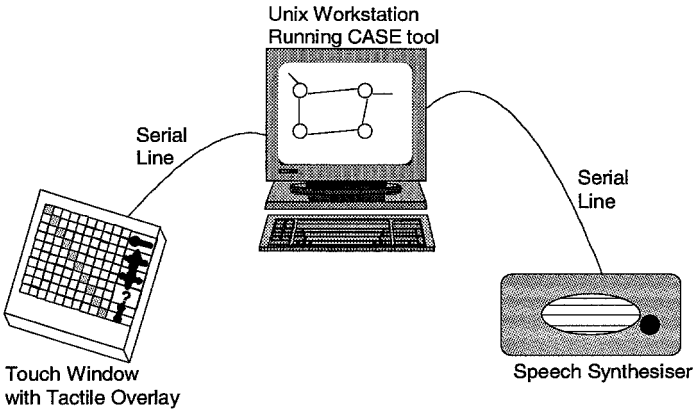

Fig. 4. The Architecture of the Evaluation System

The touch window is overlaid with a tactile diagram [10] that has the layout of a generic N^2 chart and a number of control areas to allow the user to move through the hierarchy. Thus, a blind engineer can locate significant areas on the chart by tactile means, select an area on the chart and have the associated text spoken to him/her. This is a particular instance of using "Talking Tactile Maps", which are described in [1]. The overlay is shown in Fig. 5.

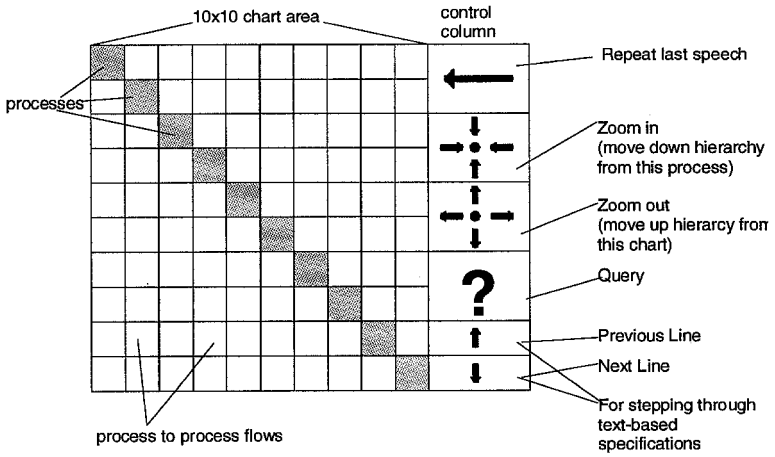

Fig. 5. Tactile Overlay for the Touch Window

The tactile overlay is divided into two parts; a generic 10x10 N^2 chart, which takes up most of the area, and a control column, which has a number of fields used by the engineer to control access to the information.

The N^2 chart encodes a DFD with the *elements* situated on the main diagonal of the matrix. To enable the engineer to find these elements these boxes are textured and are shown as being shaded in Fig.5. Depressing one of these boxes will cause

the system to speak the name of the element. For example, "*Process name is: Handle Gas Monitor.*". The other boxes represent the connection, via *flows*, of the *elements*. These boxes are not textured but the dividing areas between the boxes are raised. Depressing one of these boxes will cause the name of any connecting *flows* to be spoken; for example "*Flow name is: Water Level.*".

The control column has six fields. The first field (top right) has a horizontal arrow imprinted on it; depressing this field will cause the last text string to be repeated. This is a typical feature of talking tactile maps.

The second field is used to move down the hierarchy. Depressing an *element* square and then this field will cause the system to move to the refinement of that *element*. If the *element* is a process, this will cause either a new DFD to be accessible by using the N^2 chart or give access to the text based process specification. When entering an *element* refinement, the system speaks the name of the refined *element*, indicates the its type, and gives parameters (such as number of elements on a DFD or number of lines in a process specification). The second field is also used in conjunction with the *flow* boxes to give access to the data dictionary entry for each connected *flow*.

The third field is used to move up the hierarchy. Depressing this field will cause the current N^2 chart to be replaced by its parent and the name, type and parameters of the upper level diagram to be indicated.

The query field, indicated by a question mark, is used by the engineer to obtain information about the connections to any given *element*. Depressing this field and then an *element* square will cause the system to list all flows into and out of the *element*. This facility saves the user from having to search in every field in the matrix to determine the connectivity of an *element* and the topology of the whole diagram.

The last two fields, indicated by an up arrow and a down arrow, are used when reading textual specifications. Depressing these fields causes the next or previous line of a text specification to be read. This is used when accessing process specifications and data dictionary entries.

A 10 x 10 N^2 chart imposes a limitation on the complexity of DFDs that can be accessed by the user. This limits the total number of processes, data stores and control specifications on a DFD to 9 (the tenth entry is used for specifying interconnections with the higher level diagram). In many cases this limit of 9 is perfectly acceptable; many software engineering texts suggest that the number of processes on a diagram should not exceed 7 for it to be readable. However, in practical situations this limit of 9 will often be exceeded. When the 10 entries on the N^2 chart are exceeded, the system supports enquires about the other processes, stores and control specifications via the keyboard. If the number of additional elements is not too great, it is hoped that the chart will still be readable. The number of entries in a given N^2 chart is governed by the resolution of the Touch Window and the size of the user's finger. Larger N^2 matrices could be used if the user were to accept smaller matrix entries.

5 Conclusions and Further Work

A practical evaluation system has been developed and this will be tested by a number of blind software engineers.

Further work involves the somewhat more difficult problem of allowing blind software engineers to create Structured Analysis models. Firstly these will be supported by the engineer creating a series of N^2 charts using the Touch Window, tactile overlay and the keyboard. This will create a representation wholly in N^2 charts. As noted above, these are not commonly used in the software engineering industry and a method of creating standard, CASE tool readable DFDs, which sighted users can read, will be investigated.

Acknowledgements

The authors would like to acknowledge the work of Andrew Brook who has developed the prototype evaluation system as part of his undergraduate degree in Software Engineering from the Department of Computation at UMIST. He has contributed greatly to the ideas contained in this paper.

This project has been carried out within the Technology for Disabled People Unit at UMIST. The authors would like to acknowledge the grant funding for this unit by both The Guide Dogs for the Blind Association and the UMIST Millennium Fund.

References

1. P. Blenkhorn, D.G. Evans: A System for Reading and Producing Talking Tactile Maps and Diagrams. Ninth Annual International Conference, Technology and Persons with Disabilities, Los Angeles, March (1994)
2. G. Booch: Object-Oriented Analysis and Design With Applications. Benjamin/Cummings (1994)
3. D.J. Hatley, I.A. Pirbhai: Strategies for Real-Time System Specification. Dorset House Publishing (1987)
4. I. Jacobson: Object-Oriented Software Engineering. ACM Press (1992)
5. R. Lano: A Technique for Software and Systems Design. North-Holland Press (1979)
6. P. Loy, Y. Strapp: DFD's [sic] vs. N^2 charts. ACM Sigsoft Software Engineering Notes, 18, A16-A17 (1989)
7. J. Rumbaugh et. al.: Object-Oriented Modelling and Design. Prentice-Hall (1991)
8. P.T. Ward, S.J. Mellor: Structured Development for Real-Time Systems. 1, 2, 3, Yourdon Press (1985)
9. E. Yourdon: Modern Structured Analysis. Prentice-Hall (1989)
10. Tactual Graphics: Research and Resources. Aids and Appliances Review, The Carroll Center for the Blind, 14 (1984)
11. Teamwork/ACCESS User Manual, Hewlett Packard (1989)