South African Computer Journal
Number 6
March 1992

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 6
Maart 1992

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal

An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif

'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof John Schochot
University of the Witwatersrand
Private Bag 3
WITS 2050
Email: 035ebrse@witsvma.wits.ac.za

Production Editor
Prof G de V Smit
Department of Computer Science
University of Cape Town
Rondebosch 7700
Email: gds@cs.uct.ac.za

Subscriptions

	Annual	Single copy
Southern Africa:	R45_00	R15_00
Elsewhere	$45_00	$15_00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (int)+(012)+420-2504 Fax: (int)+(012)+43-6454 email: dkourie@rkw-risc.cs.up.ac.za

Sigma Press

Guest Contribution

This guest contribution is a slightly edited report to the Foundation for Research and Development (FRD) drawn up by Ed Coffman. Ed was an FRD-sponsored guest at the 6th South African Computer Research Symposium. The report was not originally intended for general distribution. Rather, it was specifically compiled for the FRD and its staff. I am therefore grateful to both the FRD and the author for agreeing to its publication in SACJ. I believe that it contains several incisive observations that merit further thought and discussion amongst South African computer scientists. (Editor)

Impressions of Computer Science Research in South Africa

E. G. Coffman, Jr.
AT&T Bell Laboratories, USA

In commenting on the cross section of computer science research in South Africa, I will use the classification in the table of contents of the "Summary of Awards: Fiscal Year 1989," a document published recently by the US National Science Foundation. Of the 5 categories, I will treat Numeric and Symbolic Computation as inappropriate for the discussion below. In this category I noted no research in the computer science setting in South Africa. It is also common in the US and elsewhere to place this effort in other departments, e.g., departments of mathematics or applied mathematics.

Of the remaining categories I found South Africa to be strongest in software systems and engineering, to have a substantial investment in computer systems and architecture, and to be weakest in computer and computation theory.

The coverage in software systems and engineering (SSE) was broad, topical, and similar in scope to that in US universities. Technology transfer and the corresponding relations with industry seemed to be in place or developing along promising lines. I comment in passing that this was rather surprising to me. In the US the development of SSE within university departments has lagged behind almost all other disciplines of computer science. A primary problem has been the insatiable appetite of industry for all Ph.D. graduates in the SSE field.

The investment in parallel processing, computer networks, and distributed computing appears sound, although I expected to see a greater emphasis on mathematical foundations (see my remarks below), particularly in the parallel algorithms area. Given current resources, South African institutions are doing remarkably well in computer science research. But computer science is a fundamentally important course of study, beginning at an early age and extending through graduate Ph.D. research; I take this as sufficiently obvious that I need not dwell on justifications. With this in mind, and with the necessary resources in hand, South Africa should, in my opinion, expand and consolidate its computer science research effort, increase its visibility in the international arena, and correct the rather thin distribution of graduate research among universities.

I can see much of this proceeding along present lines, but I would strongly recommend a concerted development in computer and computation theory (CCT), education and research; this is mainstream computer science and forms the basis for virtually all other fields of study within computer science. It is by no means absent in South Africa curricula, but it appears to be under--represented in advanced studies and Ph.D. level research.

At the graduate level CCT is heavily mathematical. I understand that mathematical foundations are supplied by mathematics departments in certain cases. This is not ideal, but workable and it is justified by limited resources. However, it is important that mathematics departments not regard this as a mere service; faculty will have to make a major commitment to theoretical computer science, publishing in its leading journals (e.g. SIAM Journal of Computing, Journal of the ACM, Journal of Algorithms, Algorithmica, Journal of Computer and Systems Sciences, Theoretical Computer Science, etc.), and providing the supervision of theses sponsored by computer science departments and leading to degrees in computer science. I would also encourage active participation in the international computer science "theory" societies and their meetings; two highly prestigious examples of the latter are the annual Symposium on the Theory of Computing and the Foundations of Computer Science conference.

Returning to the thin distribution of computer science research, I would make the following point. If the current situation is only a stage of development - i.e., if further resources (both human and financial) can be counted on to bring at least a few of the departments to a critical mass - then little needs to be said beyond the earlier remarks. Critical mass is hard to define, but calls for adequate, expert coverage of mainstream computer science re- search. In view of the breadth of this research, 8-10 Ph.D. full- time-equivalent faculty would seem to be barely adequate; with the usual clumping of faculty in specific research areas, more would be expected. South Africa has a talent base such that there is little doubt that such departments would achieve a much wider international recognition.

On the other hand, if resources remain fixed at current or even slightly retrenched levels, then I would recommend consolidation to achieve the same goals on a smaller scale. Within a university this can often be done by establishing interdisciplinary, degree-granting laboratories or institutes of computer science, which bring together the computer science efforts located in various departments other than computer science, such as electrical engineering, industrial engineering, business/-management science, mathematics, and operations research. The idea is to enjoy the advantages (opportunity, synergy, awareness, etc.) to both students and faculty of reasonably large computer science programs. There are many examples of such intramural laboratories in North America and Europe.

This approach could also be considered among universities within a confined geographical area, admittedly with greater difficulty perhaps. The Institute of Discrete Mathematics and Computer Science connecting Princeton University, Rutgers University, AT&T Bell Laboratories, and Bell Communications Research is a possible model. Examples in South Africa might consist of universities and research institutions on the Reef or those in the Western Cape (just to mention those with which I'm a little familiar).

As a final comment, I should note that my impressions have been based on limited information which may not give a representative picture. I am sure that my reactions will be appropriately discounted where I have been off target.

Editor's Notes

Prof John Schochot has graciously accepted to be SACJ's subeditor for papers relating to Information Systems. Authors wishing to submit papers in this general area should please contact him directly. I look forward working with John, and to a significant increase in IS contributions in future.

The hand of the new production editor, Riël Smit, will be clearly evident in this issue. Those papers not prepared in camera-ready format by the authors themselves were prepared by him in TEX. He will be announcing revised guidelines for camera-ready format in a future issue. If you use TEX or one of its variations, Riël would be happy to provide you with a styles document to SACJ format.

At last some Department of National Education committee has decided that SACJ should now be on the list of approved journals. This places it amongst the ranks of some 6800 other journals. These include not merely a number of ACM and IEEE Transactions but also such journals as *Ostrich*, *Trivium*, *Crane Bag*, *Koers*, *Mosquito News*, *Police Chief*, *Connoisseur*, *Lion and the Unicorn*, *About the House* and *Ohio Agricultural Research and Development Center Department Series ESS*. You will recall that in 1990 this same committee decided that, if judged on its own merits, SACJ did not deserve to be on the illustrious list. In the absence of other evidence, we must assume that the sole reason for its revised decision is that SACJ's predecessor, Quæstiones Informaticæ, was there. (I have a secret suspicion that the committee liked that name.)

It is my understanding that for official purposes, all journals on this list are regarded as *equally* meritorious, and all of them are more meritorious than *any* conference proceedings. What does all of this mean?

The momentous implication of the committee's deliberations is that the State will not give your institution a single cent for anything that you publish in SACJ. Instead, the State and your institution will scrupulously keep a score of the annual number of publications that count - but actually don't - because someday they might! And to encourage your enthusiastic participation in this Alice in Wonderland exercise, your institution might actually give you some of the standard subsidy funding that the State should have provided according to its own formulae, but didn't.

You will not be allowed to use this money to buy yourself a car - not even a casual meal. You may only use it to finance activities that are provably directed towards producing more papers in approved journals. The great consolation, of course, is that you will not be required to pay income tax on this money. The only tax involved will be the VAT component when you spend it in an approved manner. As a good computer scientist who enjoys recursion, my vote would be that all such revenue collected by the State should be earmarked to be placed in the pay packets of committee members who decided that SACJ should be approved.

If you publish in these approved journals with sufficient regularity and enthusiasm you will almost deserve to be regarded as a researcher. What you additionally need to do, is to ensure that you befriend and impress at least three overseas referees. You then apply to the FRD for official recognition as a researcher, and if they are sufficiently impressed, they will give you more of the non-taxable kind of money that you need to spend on research to publish in approved journals.

Derrick Kourie
Editor

Using Statecharts to Design and Specify a Direct-Manipulation User Interface

Lynette van Zijl and Deon Mitton
Institute for Applied Computer Science, University of Stellenbosch, 7600 Stellenbosch, South Africa

Abstract

Statecharts were developed by Harel et al [10] to specify complex reactive systems. In this paper we report on our application of statecharts as a design and specification tool for an X–Windows based Graphical User Interface for the telephone network performance modelling tool GMA.
CR Categories: *D.2.1, D.2.2, H.1.2, I.3.6.*
Keywords: *Human-computer interface, X-Windows, reactive systems, statecharts.*

Received October 1991, Accepted November 1991

1 Introduction

The design and specification of a user interface is often perceived as a straightforward form layout task. However, now that graphical direct-manipulation interfaces have become the norm, the complexity of the design, specification and coding of the user interface equals that of any other system component. This complexity led to a renewed interest in the specification of user interfaces over the last five years[4]. At the Institute for Applied Computer Science (ITR), where we develop commercial performance evaluation packages, we experienced the need for a modern specification methodology for direct-manipulation graphical user interfaces.

Before we discuss specification methodologies for user interfaces, let us first recapture on the structure of a typical user interface. One of the best-known models for a user interface is the Seeheim model[6]. In this model, the interface is conceived as having three components: The Presentation Component, the Dialogue Component and the Application-Interface Component (see figure 1). The Presentation Component concerns the visual aspects of the interface as presented to the user; the Dialogue Component concerns the dialogue between the user and the interface; and the Application-Interface Component concerns the connection between the application and the interface. Issues relating to the Presentation and the Application-Interface Components are less well known than the issues relating to the Dialogue Component, for which a number of specification methodologies have been proposed.

Specification methodologies can be loosely classified as being either formal or informal, and either being graphical or non-graphical. There are valid arguments for and against the choice of any of these languages; we opted for a formal graphical specification language (see section 3 for our motivations). Graphical specification languages that are based on state transition diagrams are powerful yet easy to understand and use. However, state transition diagrams suffer from two major problems when used to specify user interfaces. The first problem is the state explosion that occurs in any sizable system, and the second problem is the inability of the standard transition diagram to model concurrency. Many extensions/adaptations to state transition diagrams have been suggested in attempts to overcome these problems. One such adaptation is Harel's statecharts (see [10]).

We applied the statechart methodology in the design and specification of a large direct-manipulation interactive user interface. In our experience we found that the statecharts of Harel overcome the problems in the transition diagram specifications, in that statecharts are suited to the specification of user interfaces based on asynchronous events (such as the industry standard X-Windows system). Additionally, we found that statecharts can be applied to *design* all three of the Seeheim user interface components. As such, the statecharts provide a unified approach to interactive user interface design and specification.

The rest of this paper is organised as follows: Section 2 provides a brief introduction to the syntax and semantics of statecharts. Section 3 sketches the background to the project for which the user interface has been designed, and section 4 contains a discussion of the actual design and specification (with ample examples).

2 Statecharts Revisited

Statecharts were developed by David Harel in the mid 1980s, and are described in a number of papers[10, 12, 13]. Harel developed the statecharts as an answer to the challenge of Green[8] who claimed that there are no adequate graphical specification tools available to model interactive behaviour. Finite state machines and their transition diagrams, for example, are easy to understand, but suffer from an exponential state explosion problem.

Harel developed the generalised concept of a higraph[11] which is a diagramming object that combines the properties of hypergraphs and Venn diagrams. The higraph can therefore be used to represent a set of elements (with a special relation on them) and then to represent a collection of such sets with structural relationships among them. The interested reader can consult [11] for more in-

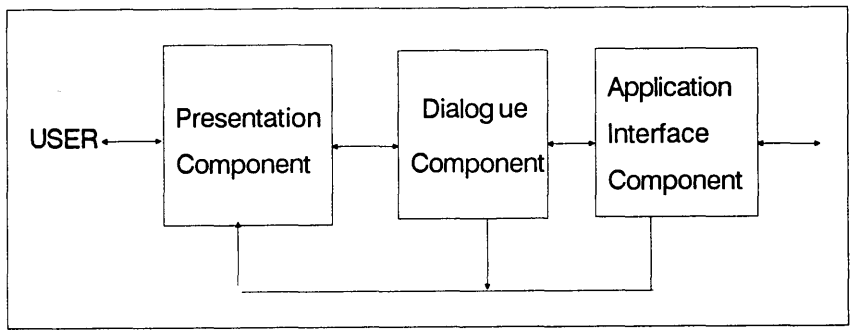


Figure 1: The Seeheim User Interface Model

formation concerning higraphs; let it suffice at the moment to note that the higraph is a formally defined mathematical object, with a visual representation of its topological characteristics. The statechart is just an application of a higraph; it is essentially a higraph-based version of a finite state machine.

Since the statechart is based on a finite state machine, it is empowered with all the ergodicity of a state transition diagram. Moreover, it solves the state explosion problem found in finite state machines by having the properties of depth and orthogonality. The depth property allows for hierarchical modelling by clustering or refinement, so that atomic elements can be clustered graphically into a single composite element, or a composite element graphically refined into several smaller elements. The orthogonality property can be used to model independence and concurrency, by combining elements graphically in a representation of their Cartesian product.

Before illuminating these concepts with an example, let us first highlight the graphical symbols used to construct a statechart. Statecharts graphically represent the states and events in a system. The symbol used for a state is a rounded rectangle, with the name of that state inside the rectangle. Hierarchies of states are represented by enclosure of states inside outer states (see figure 2). Events are represented by arrows, with the arrow labelled by the name of the event and any condition which may guard that event. There are different types of arrows available. The standard arrow is that which originates on the perimeter of one state and ends on the perimeter of another state (see *Event-1* in figure 3). The default entry state is labelled by a default arrow (such as *Event-0* in figure 3). The hierarchical arrow indicates that an event originates in a lower level state inside the current state. For example, in figure 3 *State1* is an hierarchical state with substates on a lower level. Here *Event-2* originates from one of the states within *State1* and not from *State1* as such.

Now that we have the symbols for a statechart available, let us use them to model a small example. Suppose that the system to be specified is a chocolate vending machine called C (the original vending machine idea is due to Hoare[14]). The vending machine is always willing to accept money, irrespective of whether it has the chocolates to uphold its pledge! However, it is at least honest – if it has no more chocolates to exchange for the money, it returns your coin. It is also a fairly clever vending machine; the slot in the machine allows only the correctly sized coin to be inserted, and other objects entered by dishonest or hungry customers are simply ignored. The first step in the statechart specification is to identify the states and the events in the system. To the outside observer, there are two visible states: The *READY* state in which the machine is willing to accept money, and the *BUSY* state in which it processes a request for a chocolate. There are three relevant events: The event of a coin insertion into the machine, the event of a chocolate returned to the customer, and the event of a coin returned to the customer. This can be represented in a statechart as depicted in figure 4. Note the different types of arrows used to represent the different types of events.

One can now continue to refine the system C. Clearly, there must be some busy state in which C has chocolates to return to the customer, and some busy state in which no chocolates are available. Note in figure 5 how the enclosure of the states *NO-CHOCS* and *MORE-CHOCS* in the state *BUSY* indicates that only one of those states can hold at any one time (that is, encapsulation enforces an XOR).

It now remains to specify the event(s) which lead to the states *NO-CHOCS* and *MORE-CHOCS* from the *READY* state. Obviously, it is the same event in both cases, namely, the event *coin-accepted*. Since the same event leads to all the states enclosed in the *BUSY* state, we can use one arrow ending at the perimeter of the *BUSY* state (see figure 6).

Suppose now that there is a vending machine with exactly the same functionality as C, but which dispenses toffees. Let us call this machine T. We can now design a vending machine called CT from the specifications of C and T, which operates as follows: There are two slots for entering coins, one for a toffee and one for a chocolate (the price is the same). Suppose that a customer wants to buy a toffee and enters a coin in that slot. As before, if there are any toffees available, the machine delivers a toffee. However, if there are no toffees available, CT first checks whether there are any chocolates available. If there are, it returns a chocolate instead of a toffee. If there are neither toffees nor chocolates available, it returns the customer's coin. A similar return policy applies if a customer requests a chocolate. Graphically CT can be specified by a statechart as depicted in figure 7. Note that the dashed line indicates

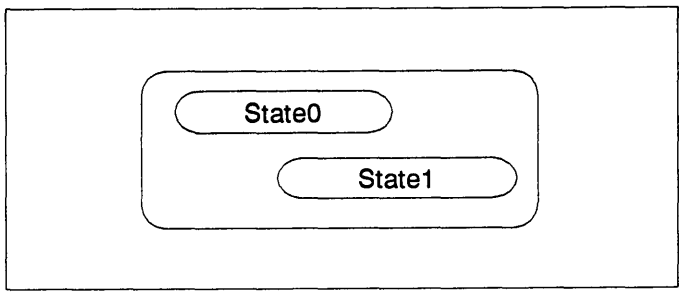


Figure 2: Hierarchies of states

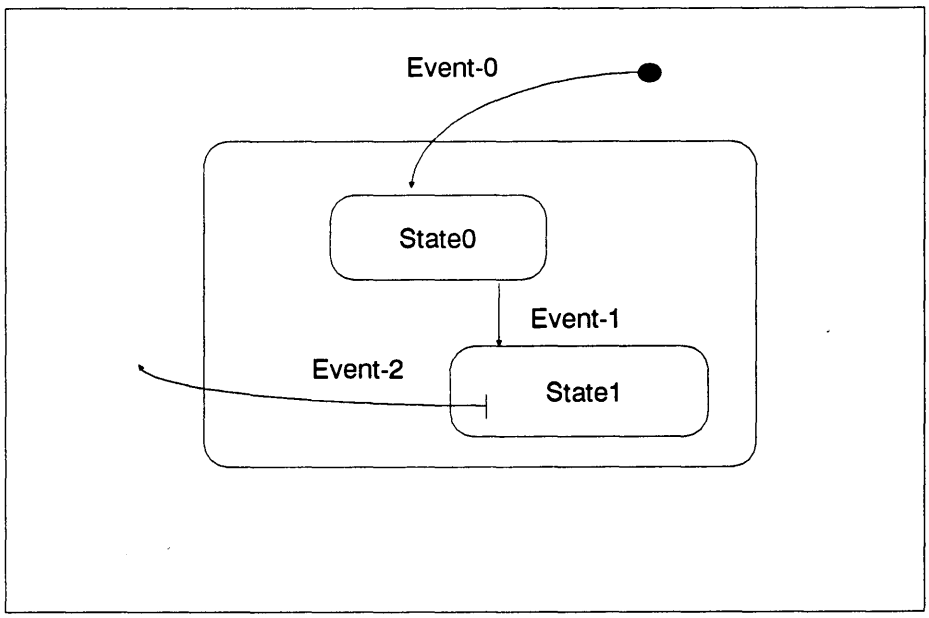


Figure 3: Events in a statechart

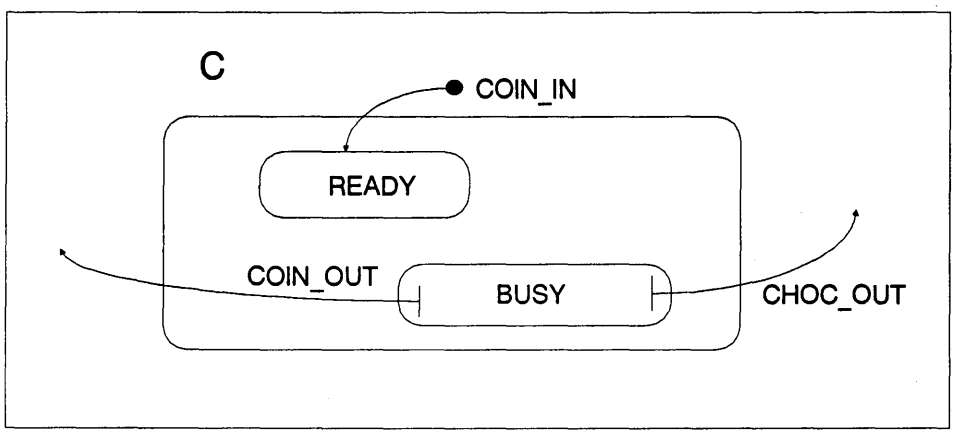


Figure 4: Chocolate Vending Machine Stage 1

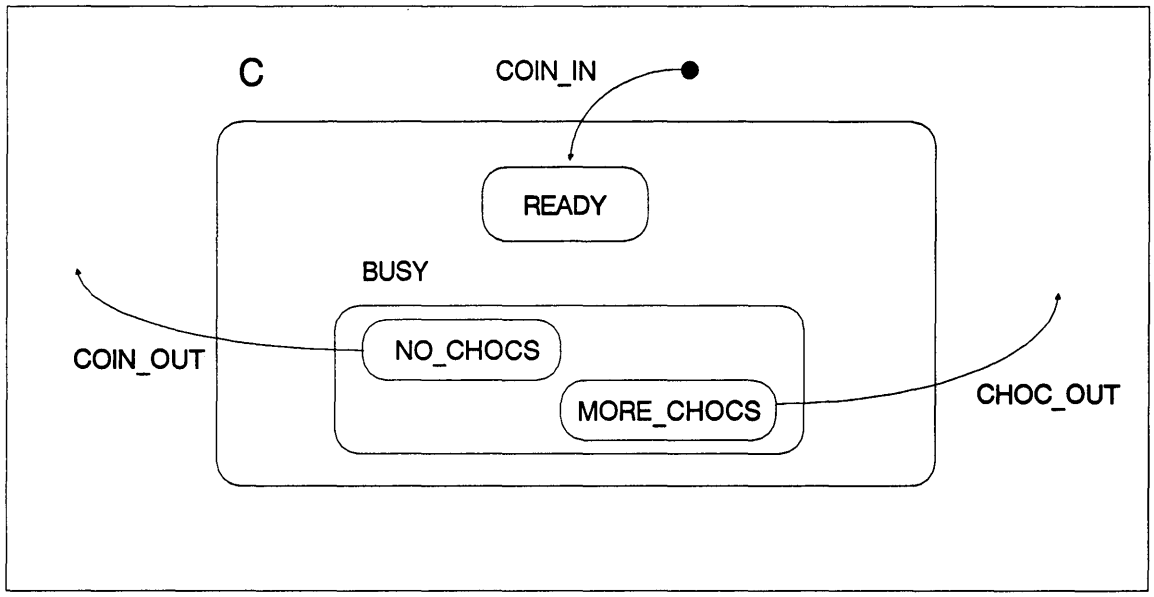


Figure 5: Chocolate Vending Machine Stage 2

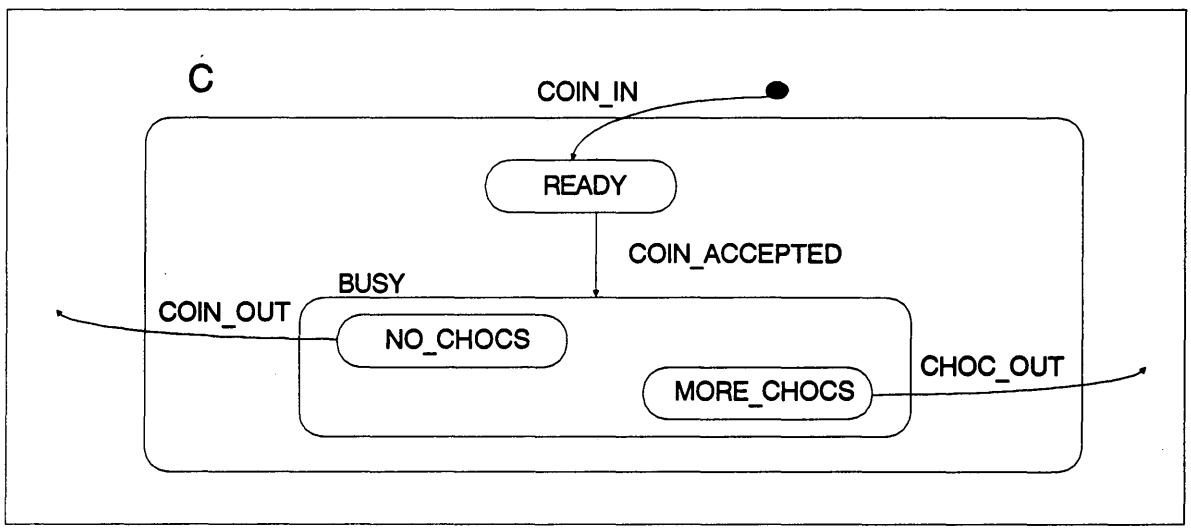


Figure 6: Chocolate Vending Machine

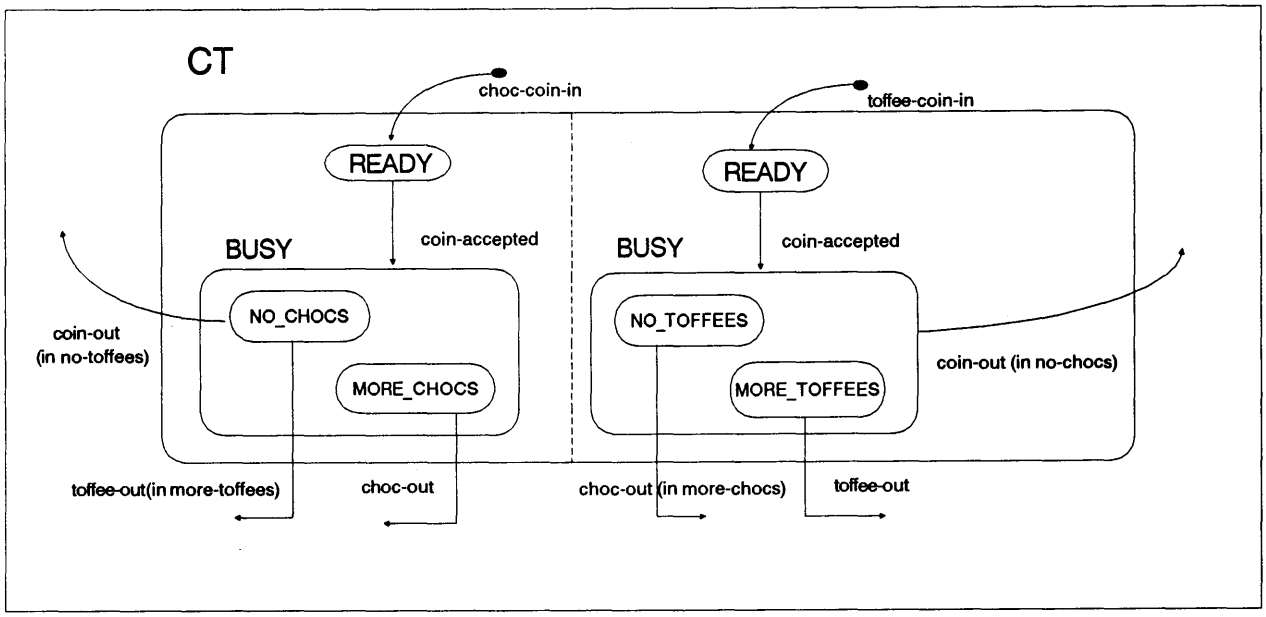


Figure 7: Chocolate and Toffee Vending Machine

the Cartesian product of the two machines, so that it is in principle possible to simultaneously enter a coin for a toffee and a coin for a chocolate. The dashed line thus enforces an AND condition for the states in the system.

Another useful modelling mechanism is the selection arrow. The selection arrow is indicated by a circled S on the arrowhead of an event arrow. It can be interpreted as a generic event, with a number of clearly defined values to indicate which state is triggered by the event. For example in a menu driven system, this mechanism can be used to indicate the event of selecting one of many possible menu items. Detailed examples using this mechanism are given in section 4.

The explanation above covers only those aspects of statecharts that we need to design and specify the GMA example in section 4. The reader is again urged to consult the original papers for more information.

3 The Graphical UI for GMA

The GMA (Graphical Modelling and Analysis) package is a performance modelling tool for telecommunications networks such as the South African public data network Saponet–P and the South African telephone network. The package has already been described in other papers ([3, 17]) and this section provides a summary of the overall project followed by more detail on the aspects relevant to this paper.

The GMA package runs under Unix, with graphics based on the X-Windows system using the X Toolkit and OSF/Motif. The structure of the GMA package is shown diagrammatically in figure 8. The package consists of three independent modular units: The Data Extraction Software, the Model Solution Kernel and the Graphical User Interface. Both the data extraction software and the model solution kernel are independent of the graphical interface; a standard interface to each module serves to hide its implementation details. This facilitates one of the main advantages of the package, namely its utility in modelling diverse computer network implementations. Such diverse modelling is achieved by modifying the front-end modules to cater for installation-specific features. The graphical interface controls the extraction of configuration and traffic data from the communication network under study. It presents a diagram of the network to the user and constructs the performance model of the network. The model is solved by the solution kernel, and results are presented graphically on the diagram.

One of the main goals of the GMA project was to produce a modelling package which could be used by network engineers with no training in the construction of statistical performance evaluation models. This implies that we had to design an interface that would present a graphical view of the network in a familiar form. This presented no problems in the case of data communication networks, as GMA could simply present the user with a geographical and/or a schematic view of the network nodes. The user can then switch between the geographical and schematic view, or zoom into a specific node of the network (see figure 9).

The interface for the telephone network did however present immediate problems. Typical telephone networks are hierarchical in nature, with calls routed among the different levels, as depicted in figure 10. Moreover, for historical reasons the typical telephone network actually consists of various overlay networks, with the different networks connected by crossover points (see figure 11). In addition, the characteristics of the network differ depending on geographical location – for example, the trunking and routing decisions for metropolitan areas differ from that of rural areas. Telephone networks also contain many more

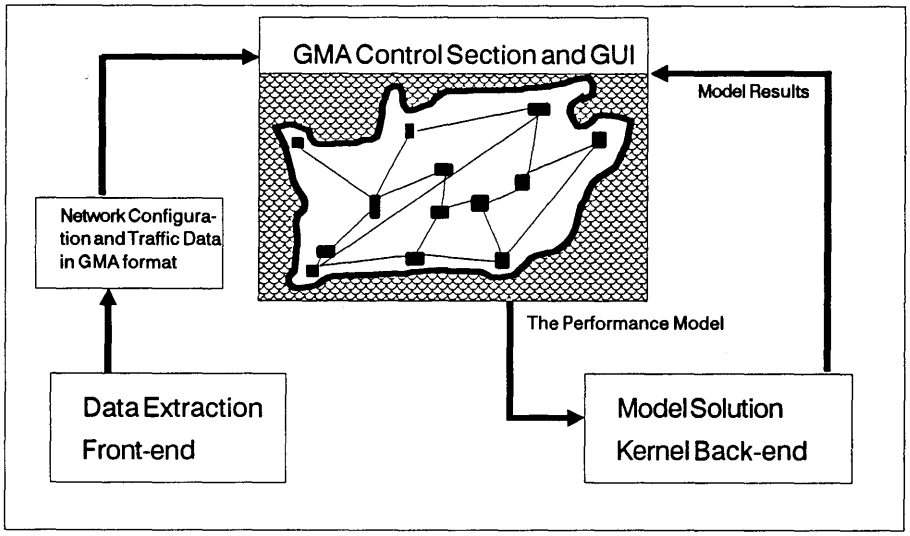
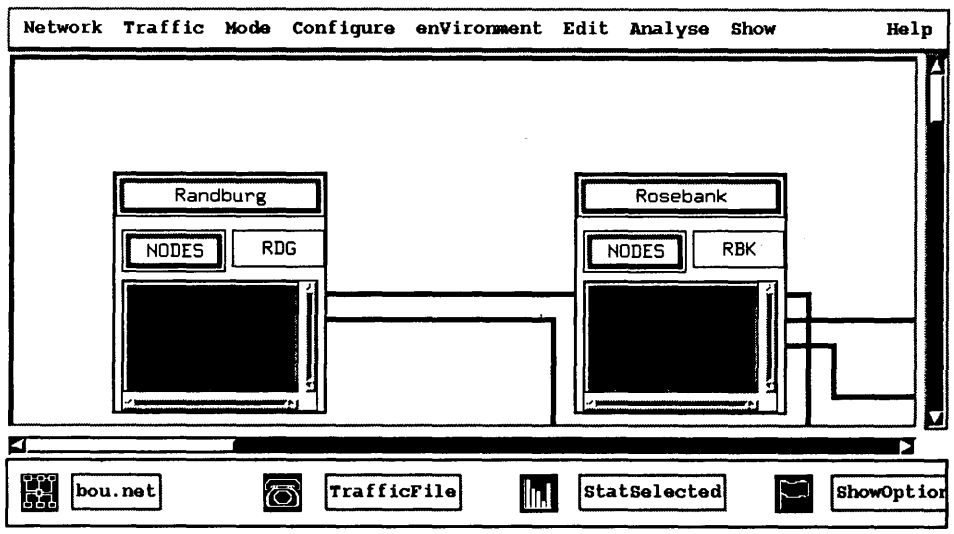


Figure 8: The Structure of the GMA Modelling Tool



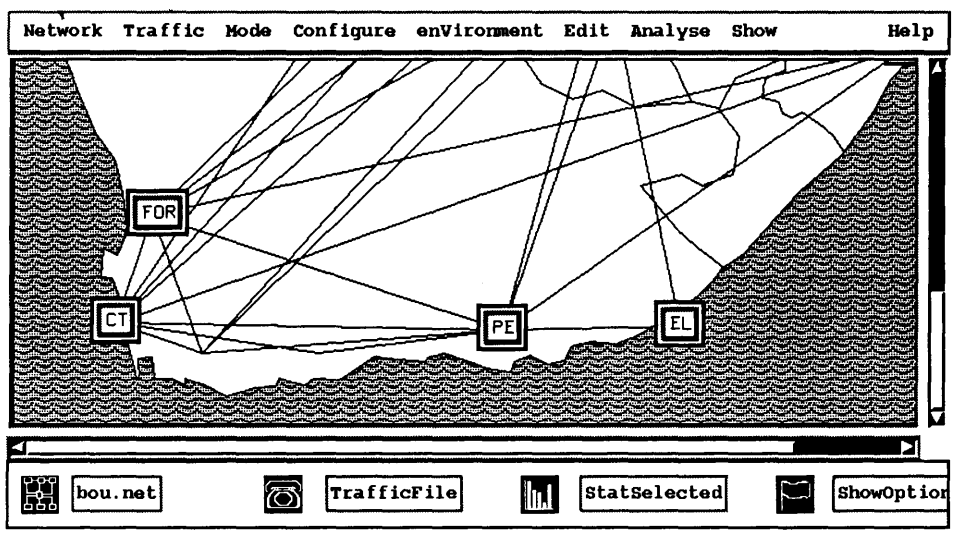


Figure 9: The Schematic and Geographical Modes in GMA

nodes than the typical data communication network. For example, the South African telephone network contains approximately 1100 nodes, while Saponet–P contains 31 nodes.

A graphical interface for the modelling of a telephone network thus has to take all the above factors into account. The user should be presented with a hierarchical outline view, a view of only one level of the hierarchy, and localized geographical area views. A facility must also be available to zoom in on the small representation of a node, the small representation being necessitated by the large number of nodes in the network. It should be possible to switch between representation modes without affecting the network information known to the system.

A user-friendly interface for the telephone network modelling in GMA is therefore clearly a complex system. The design of such an interface has to take into account all the events possible in any one mode, and the effects of such events. We found a natural language specification of such an interface to be open to too much ambiguity, and started a search for a formal design and specification tool suited to our needs. Axiomatic (for example, [2]) and other textual specifications (such as BNF, see [15]) were discarded, because they lead to specifications that are difficult to read and maintain for large complex interfaces such as the GMA interface. We thus set out to find a graphical specification method whereby we could design and specify an event-driven interactive user interface. Specification systems without a formal underlying methodology (see [9]) were discarded, mainly because of our interest in the provable correctness of the GMA system ([19] discusses this issue). The need for a formal graphical specification method pointed to the use of some extension or variation of state transition diagrams, such as that of Wasserman [20], Zave[21] or Harel[10]. Wasserman's USE methodology uses ATN's (Augmented Transition Nets), and suffer from the state explosion problem. Although Zave's sequence diagrams have the same expressive power as statecharts, we found the explicit expression on concurrency easier to understand and use. We thus set out to design and specify the GMA interface using statecharts.

4 The Design and Specification of the GMA Interface

In this section we show how the GMA interface was progressively designed and specified using statecharts. Throughout the discussion we point out how statecharts can be used to design all three the components of the user interface, and we comment on the ease of use of its concurrency specification mechanism. We shall follow a standard procedure in the design and specification of the GMA user interface. We always start with the design of the first component of the Seeheim model (i.e. the Presentation Component) and given the Presentation design, we proceed to specify the Dialogue Component in more detail. The Application Interface Component can then be designed using the information about the external events in the Dialogue Component.

The GMA package should present the user with a logo on startup time, after which the proper working screen should be displayed. So, a global view of the overall system can be specified as in figure 12 where there are two non-concurrent states: The *LOGO* state and the *GMA-PROPER* state. Note how the design for the actual presentation of the screens (the Seeheim Presentation Component) is visible from the statechart: the XOR of the two states *LOGO* and *GMA-PROPER* with the given events implies that those two windows/screens cannot be visible simultaneously. The default entry point to the system is the LOGO state, through the external event *gma*. The event *ok* places the system in the *GMA-PROPER* state, from which the event *CTRL-Q* will cause a quit from the system. Since there is no event back from *GMA-PROPER* to *LOGO*, it is not possible to switch between these two visual presentations. The *ok* event together with the *LOGO* and *GMA-PROPER* states define the Dialogue Component, while the external events *gma* and *CTRL-Q* define the Application Interface Component.

Let us look at the state *GMA-PROPER* in more detail. The visual layout of the window must comply with the OSF/Motif style guide and our house rules on screen layout and as such consist of three display areas: A Status Display Area, a Menubar and a Working Area. All three these areas should be visible simultaneously. The window layout (Presentation Component) can for example be presented as in figure 13. The corresponding statechart, with the relevant events, is given in figure 14. Note that, although the design of the Presentation Component of the user interface is clearly only informal, the statecharts do allow the designer to form a preliminary picture of the layout of the window as the design progresses. As before, the design of the Application-Interface Component can be derived from the (only) external event *ok*.

Each of the three display areas can now be specified separately. The Status Display Area is the simplest of the three in terms of the possible events. It is an example of a display area where the user cannot enter any information but the application can set or display status information. The events that can be input to the Status Display Area are all of the same generic type (i.e. set one of the status flags) and we use the selection arrow to specify these events (see figure 15). Note that 'illegal' events are ignored by the statusbar, in that there is simply no reaction.

The second area of the *GMA-PROPER* window is the Menubar. The Menubar Area consists of a constantly visible menu bar, drop-down menus and pop-up selection boxes. Again, the selection of a menu item from the menu bar is specified using a selection arrow. Notice how an event can be aliased to hide implementation detail: The *OptionSelected* event can be implemented to allow multiple methods to select an option, such as a click on the mouse pointer and keyboard input (see figure 16). The *MenuBar* state itself is simple to specify, as illustrated in figure 17.

For the state *DropdownMenus*, a generic dropdown menu is specified to illustrate the applicability of statecharts to this menu type. Consider any drop-down menu system with n dropdown menus. Each menu is either vis-

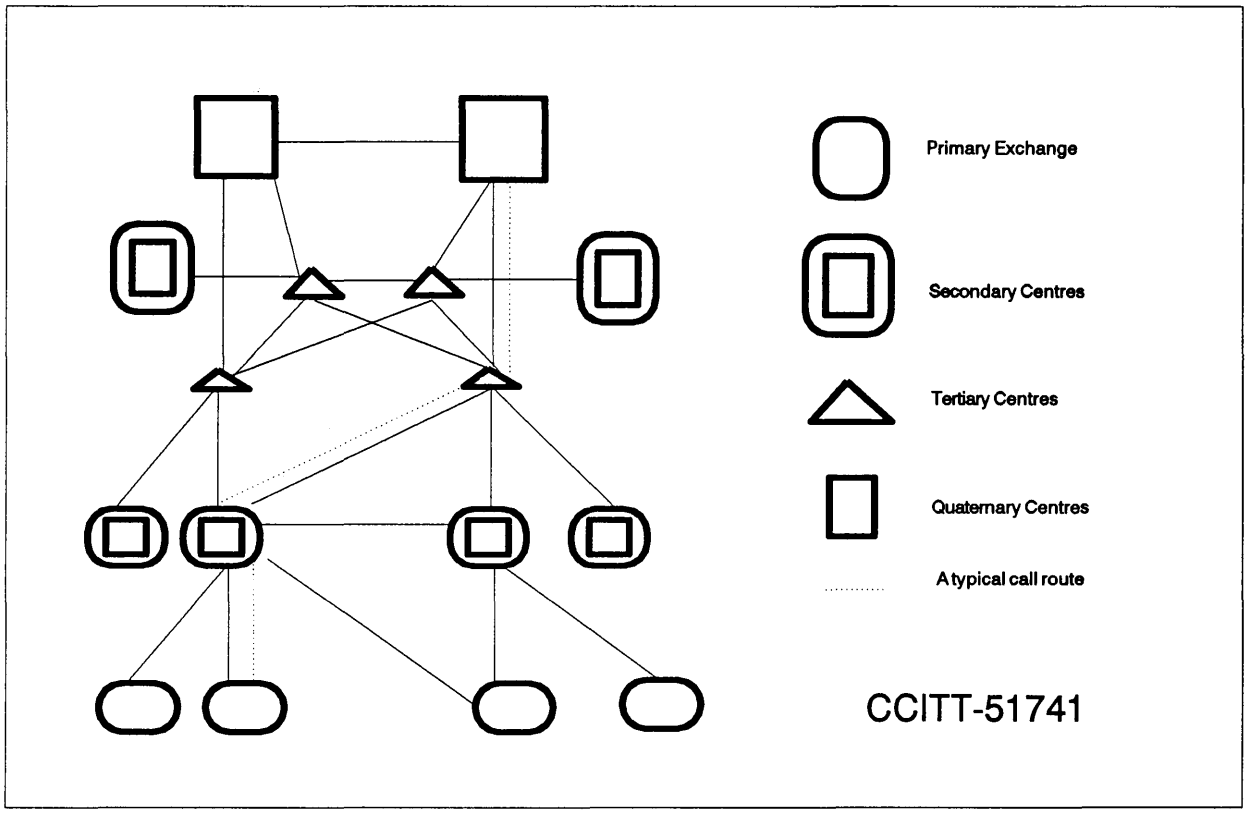


Figure 10: The Hierarchical Levels in a Telephone Network

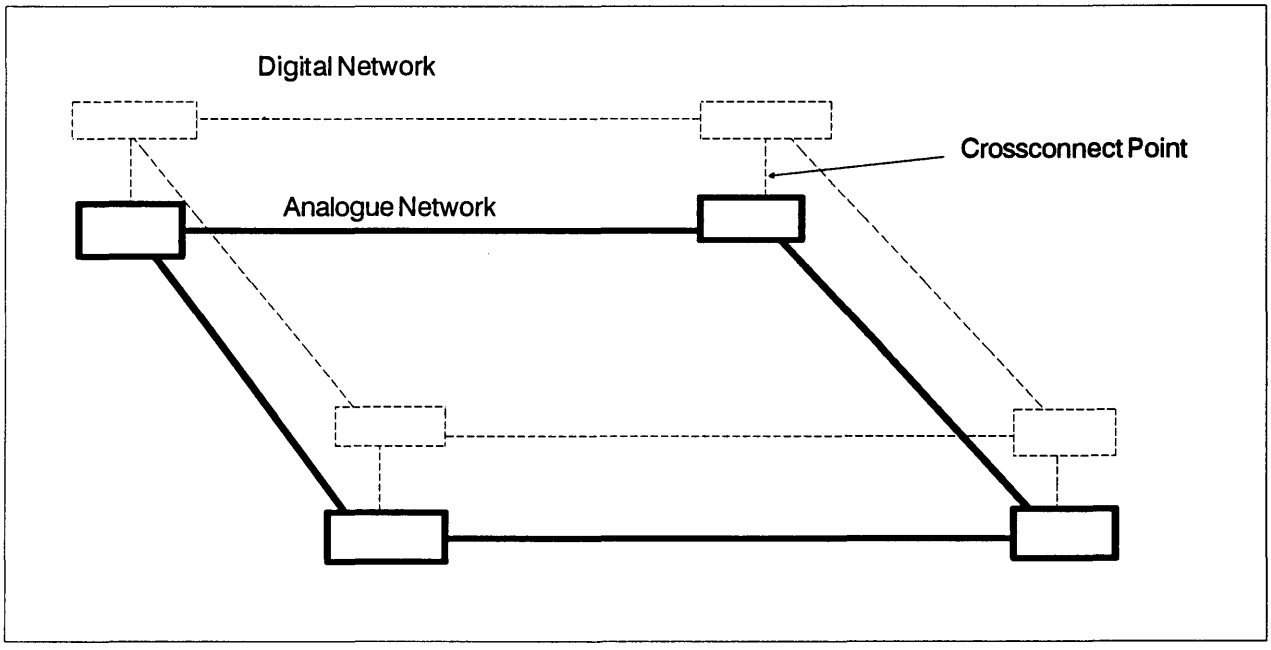


Figure 11: Two Overlay Networks in a Telephone Network

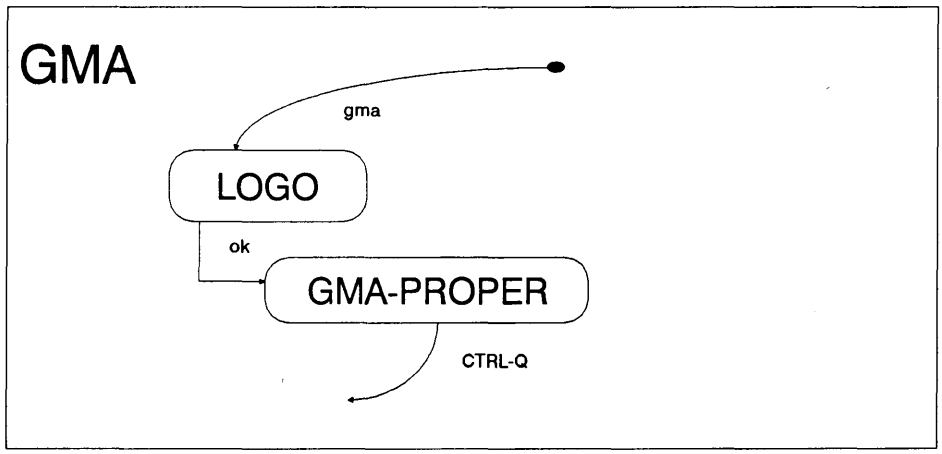


Figure 12: A global view of the GMA system

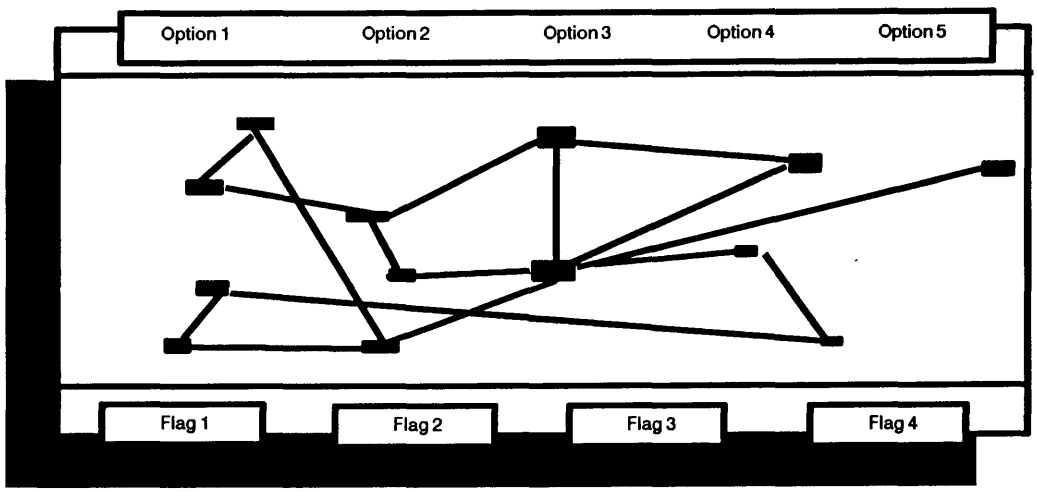


Figure 13: The Main Window Layout for GMA

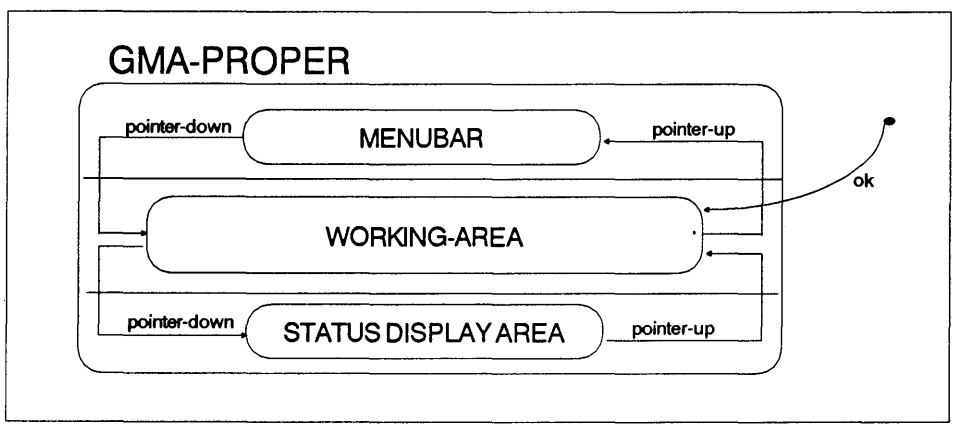


Figure 14: GMA-PROPER

ible (when it is selected) or not visible. However, more than one menu cannot be visible simultaneously. The user can move between menus by dragging the pointer to the left or the right, or by exiting a menu and selecting a new option (on the menu bar). The statechart that models state *DropdownMenus* is depicted in figure 18.

When a dropdown menu is visible, a number of options are displayed, with the pointer indicating the current (highlighted) option. Moving the pointer will switch between options, and clicking on an option selects that option. Alternatively, one can use mnemonics or accellerators from the keyboard to choose any of the options in that particular drop-down menu. Figure 19 contains the statechart for the visible state of a drop-down menu.

As a typical example of a popup menu, we selected to design and specify a Threshold Editor box. In the GMA context objects are colour coded to indicate their 'safety' level according to a certain threshold. The system has default numerical values for all thresholds, but the user is allowed to change them with the Threshold Editor. The editing should occur graphically and/or numerically, and the changes in the values should be visible in colour and in numerical values. We use a scale with three different colours, where the boundaries of the coloured areas of the scale may be dragged to edit the thresholds. As usual in graphical user interface applications, three standard buttons are provided to commit the changes, cancel any changes, or to provide help. The Presentation Component of this popup menu is given in figure 20, with the corresponding statechart in figure 21.

The easiest specification in the Threshold Editor is the *BUTTONS* state. The mouse pointer may be in the buttons area of the Threshold Editor box, without being on any of the buttons. In that case, none of the buttons are selectable. We call this state the *IDLE* state. If the mouse pointer is on any of the buttons, then the buttons are potentially selectable. We call this state, for each button, its *ACTIVE* state. When the mouse pointer is clicked on a button, its corresponding reaction is activated. This is the *SELECTED* state for each button. So, the statechart looks like figure 22.

The statechart for the *BUTTONS* state in figure 22 is a clear example of how the Application-Interface Component can be designed with statecharts. The external events from each of the *SELECTED* states of the buttons defines the (in X-Windows terminology) 'hook' between the interface and the application.

The *NUMERALS* and *SCALE* states can be specified in a straightforward fashion, reminiscent of the statechart for the Status Display Area in figure 15. The statecharts are given in figure 23.

Up to this point, we have illustrated how to design some standard graphical user interface features using statecharts. However, the third area of the *GMA-PROPER* state is the Working Area, which consists mostly of application-specific graphics as opposed to the general menu system. We will now show how the statecharts measured up to their potential in the design of the Working Area.

As explained in section 3, the Working Area must present to the user of the system both a geographical and a schematic view of the network. Additionally, the user must be able to zoom in on a specific geographical area, and switch between geographical and schematic views as required. Moreover the user should be able to select the hierarchy of the network currently visible (refer again to figure 10). The statechart to model these requirements are given in figure 24.

Each of the states in figure 24 can now be refined into a more detailed statechart. For example, let us consider the *AREA-GEOGRAPHICAL* state (see figure 25). Upon state entrance, a selection determines which geographical area should be displayed. The corresponding set of nodes are displayed (the *NODE* state indicates the graphical display of a node with its corresponding links). For any of these nodes, the node can be active or selected. Note again that, as in the Threshold Editor Box, the system can be in an idle state. Further refinements can now be designed according to the action taken upon node selection.

The reader will notice that, in the more detailed statecharts, how the user interactions are modelled by the events among states. In the implementation of these statechart designs, these events usually specify the third component of the Seeheim model, namely, the Application–Interface Component.

This completes the detail of our design of the GMA direct-manipulation user interface with statecharts.

We believe that this section supports our claim on the superiority of statecharts as a specification mechanism for interactive event-driven systems.

5 Conclusion

In this paper we set out to show that statecharts are well-suited to the formal specification of event-driven user interfaces based on X. We believe that this claim is substantiated by the empirical results illustrated in section 4. In that section we discussed the design of a non-trivial direct-manipulation graphical user interface, and showed how the statecharts can be applied to design all three components of the Seeheim model. In summary, the main advantages of the statechart approach in our experience are the following:

- The statechart itself has been fully formalized, increasing the feasibility of its use in systems where formal correctness is an issue.
- Of the graphical design methodologies that we considered, the statecharts provide the easiest and most natural way to model concurrency. This is essential in the case of interfaces based on X-Windows.
- The statechart design can proceed hierarchical, ensuring that the designer can visually modularize the design so that the detail required at any stage is minimal.
- The statechart is a natural formalism for most programmers, so that the learning curve for the methodology is remarkably short.
- All the components of the Seeheim user interface model can be designed with the statecharts, providing a uniform approach to the design and specification of the user interface.

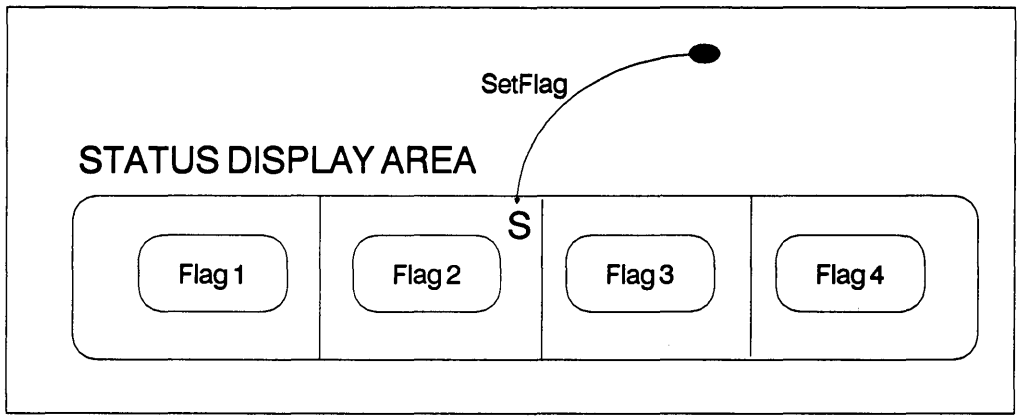


Figure 15: The Status Display Area

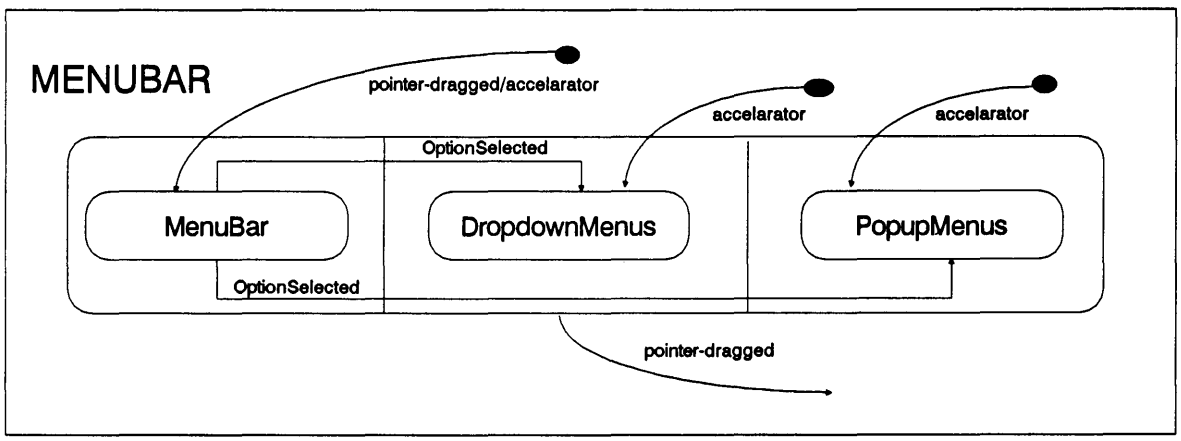


Figure 16: The Menubar Area

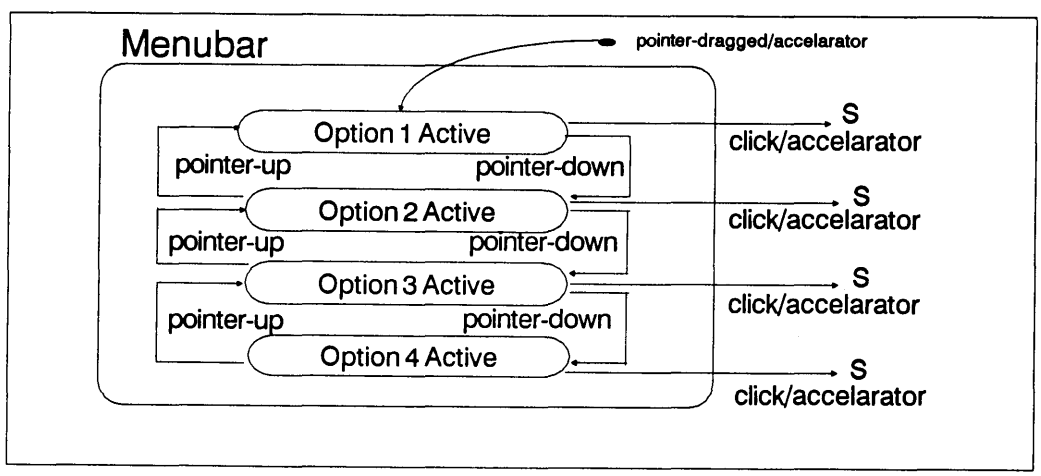


Figure 17: The MenuBar State

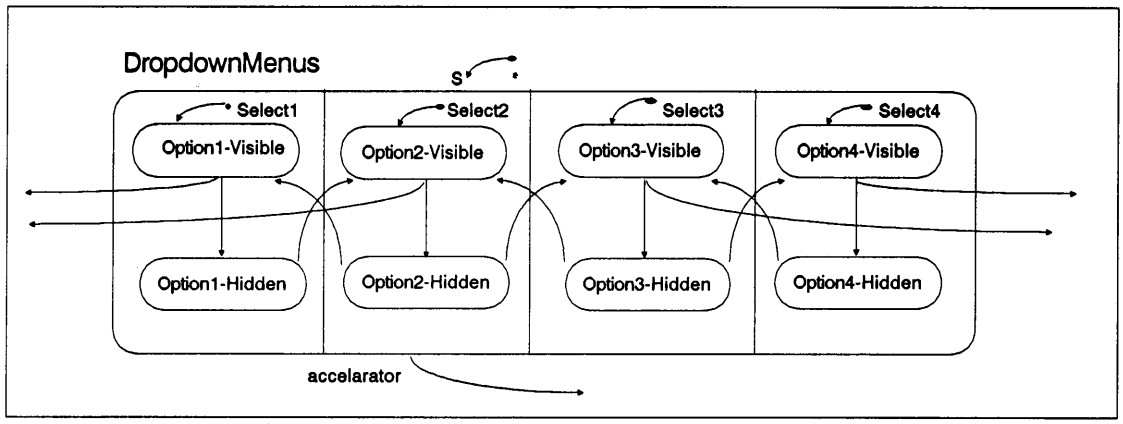


Figure 18: The DropdownMenus State

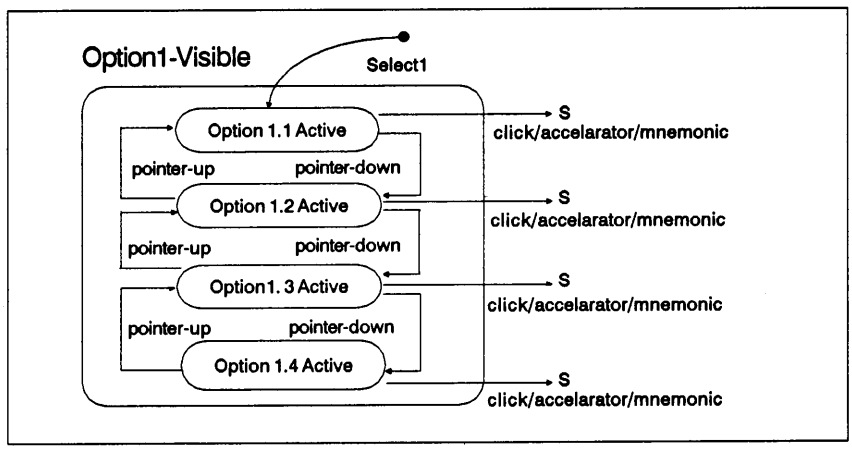


Figure 19: A Visible Drop-down Menu

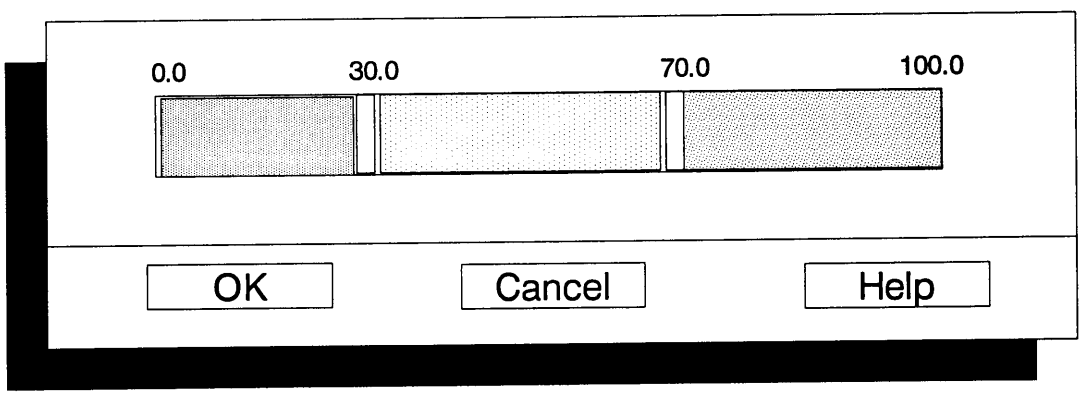


Figure 20: The Layout for the Threshold Editor

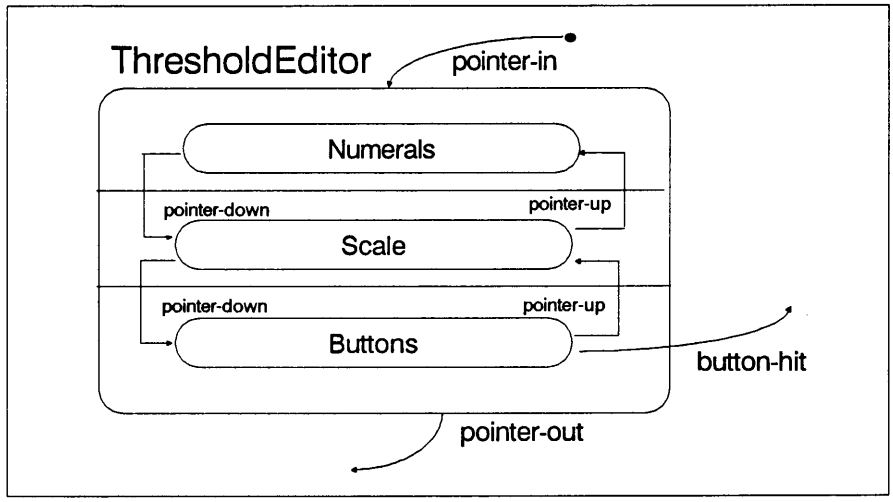


Figure 21: The Statechart for the Threshold Editor

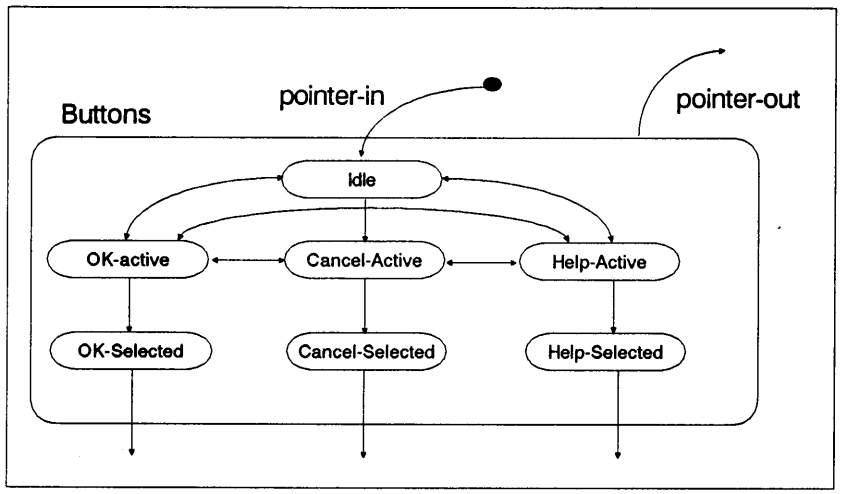


Figure 22: The Buttons Statechart

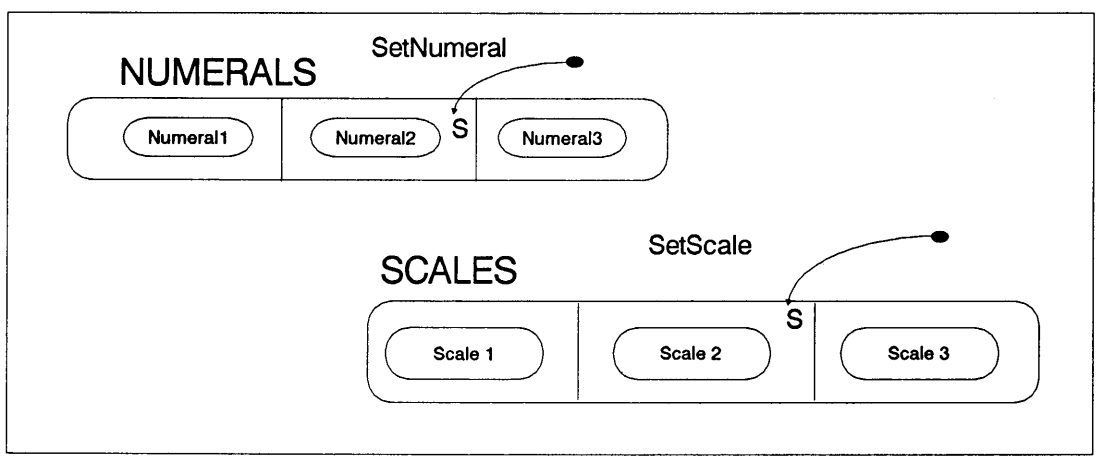


Figure 23: The Numerals and Scales Statecharts

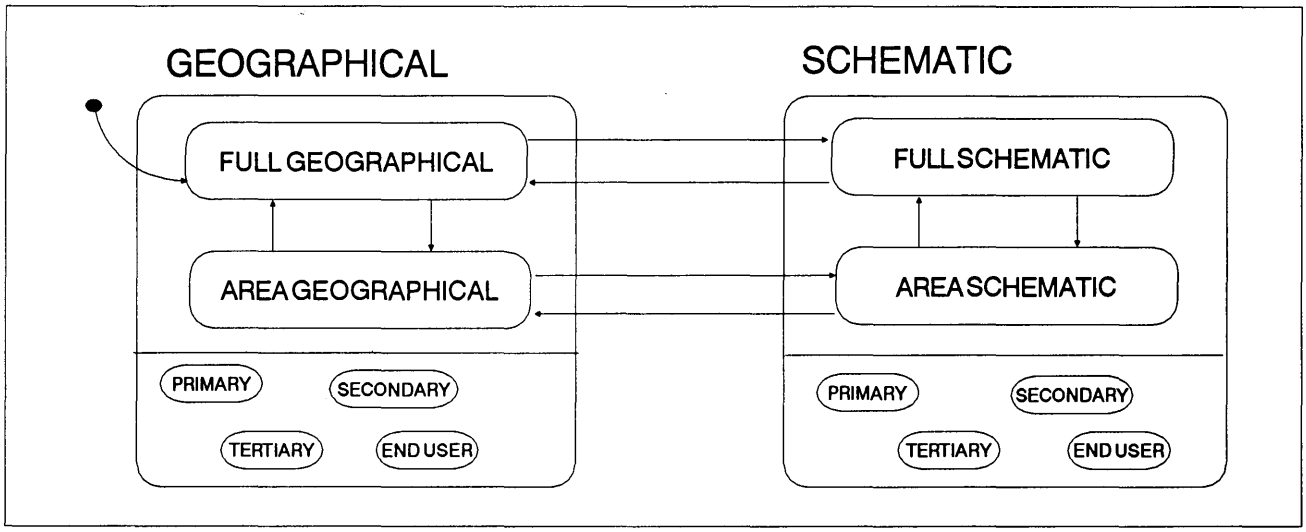


Figure 24: The GMA Working Area Statecharts

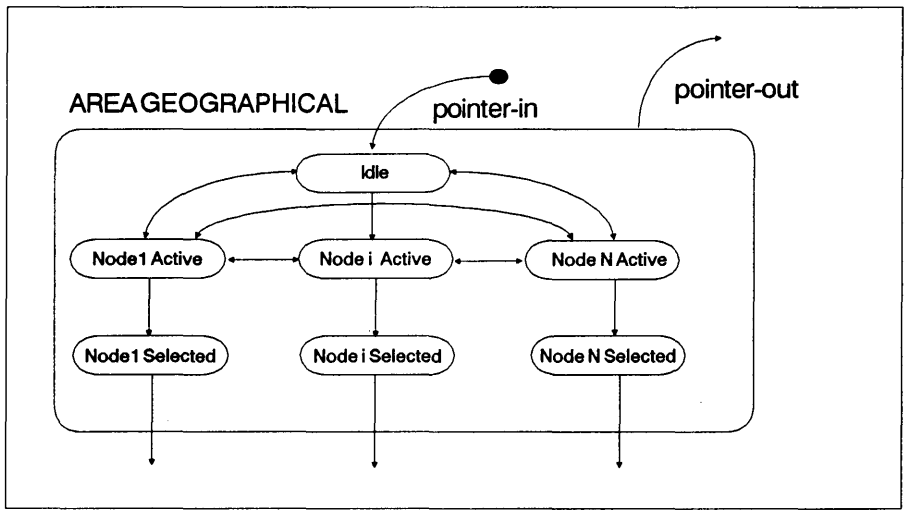


Figure 25: The Area Geographical Statechart

One of the only disadvantages of the statechart approach that we encountered, was that (specifically in object-orientated user interfaces) the state-based approach was not always the most natural approach to describe graphical objects. However, after a few examples we found that we could exploit the generic similarity between graphical object behaviour, so that most of our designs are adaptions of previous designs. This is in fact a hidden advantage, since one can thereby ensure that the behaviour of the interface is constant throughout the system.

References

[1] Bass LJ, An Approach to User Specification of Interactive Display Interfaces. IEEE Transactions on Software Engineering SE-11 (8), August 1985, pp 686–698.

[2] Chi UH, Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches. IEEE Transactions on Software Engineering SE-11 (8), August 1985, pp 671–685.

[3] Crosby S *et al*, A Graphical Tool for the Modelling of Packet and Circuit Switched Communication Networks. Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Turin, February 1991.

[4] Foley J, Guest Editor's Introduction: Special Issue on User Interface Software. ACM Transactions on Graphics 5:2, April 1986, pp 75–78.

[5] Gait J, An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows. IEEE Transactions on Software Engineering SE-11 (8), August 1985, pp 714–717.

[6] Green M, Report on Dialogue Specification Tools. User Interface Management Systems, Proceedings of the Workshop on UIMS, Seeheim, Federal Republic of Germany. Springer-Verlag, January 1985.

[7] Green M, A Survey of Three Dialogue Models. ACM Transactions on Graphics 5(3), July 1986, pp 244–275.

[8] Green TRG, Pictures of programs and other processes, or how to do things with lines. Behavior Inform. Tech. 1(1982), pp 3–36.

[9] Harbert A, Lively W and Sheppard S, A Graphical Specification System for User-Interface Design. IEEE Software July 1990, pp 12–20.

[10] Harel D, Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8 (1987) pp 231–274.

[11] Harel D, On Visual Formalisms. Communications of the ACM 31(5), 1988, pp 514–530.

[12] Harel D *et al*, STATEMATE: A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering SE-16 (4), April 1990, pp 403–413.

[13] Harel D *et al*, On the formal semantics of statecharts. In Proceedings of the 2nd IEEE Symposium on Logic in Computer Science (Ithaca, N.Y.,June 22–24). IEEE Press, New York, 1987, pp 54–64.

[14] Hoare CAR, Communicating Sequential Processes. Prentice-Hall, 1985.

[15] Jacob RJK, Using Formal Specifications in the Design of a Human-Computer Interface. Communications of the ACM 26(4), April 1983, pp 259–264.

[16] Jacob RJK, A Specification Language for Direct-Manipulation User Interfaces. ACM Transactions on Graphics 5(4), October 1986, pp 283–317.

[17] Mitton D *et al*, The Development of a Large Graphical Interface using X and the X Toolkit. Proceedings of the SAIEE Software Engineering Professional Group Summer Meeting, Johannesburg, January 1991.

[18] Myers BA, Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints. ACM Transactions on Programming Languages and Systems 12(2), April 1990, pp 143–177.

[19] Van Zijl L, Visual Verification. Proceedings of the SAIEE Autumn Meeting, 11th April 1991, Johannesburg.

[20] Wasserman AI, Extending State Transition Diagrams for the Specification of Human-Computer Interaction. IEEE Transactions on Software Engineering SE-11 (8), August 1985, pp 699–713.

[21] Zave P, A Distributed Alternative to Finite-State-Machine Specifications. ACM Transactions on Programming Languages and Systems 7(1), January 1985, pp 10–36.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages*, McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, $\TeX$ or $\LaTeX$ or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, $\TeX$ or $\LaTeX$ documents.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 6, March 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 6, Maart 1992
ISSN 1015-7999

Contents