Regardless of the size of your computer system it is a strategic tool. We are committed to helping you use it most effectively. All it needs is a little imagination.

UNIDATA
Telephone (012) 45-1152

THERE'S A GIANT IN EVERY ONE OF US
The Department of Computer Science at Pretoria University is inviting applications for two annually awarded bursaries for MSc study. These bursaries are aimed at outstanding students who have a Computer Science Honours degree (or equivalent qualification). As far as possible, one such bursary will be awarded to a South African student, while the other will be awarded to a student from another African state. The hope is to thereby promote interaction between South African computer scientists and computer scientists elsewhere in Africa.

The bursaries are funded by the Department’s contract research work and will amount to a minimum of R20000 per year. They are intended to cover costs for tuition, housing, transport, books, and an adequate allowance for other living expenses. Successful candidates will be required to work for 20 hours per week as part of the Computer Science Department’s contract research team. The work assigned to each candidate will be aimed at providing him with practical experience in his field of interest.

Study fields for which bursaries will be considered include:
- Computer Networking
- Distributed Systems
- Software Engineering
- Artificial Intelligence
- Intelligent Computer Aided Instruction
- Machine Aided Translation

Students should aim to complete the dissertation that is required for an MSc within two years of study (i.e. one year of full time study). The dissertation should preferably be written in English but it may also be written in Afrikaans.

The continuation of the bursary and adjustments to the allowance will be decided upon annually and will depend on progress. It is a condition of the bursary that, after completing his/her studies, the student should return to his/her country of origin for the same number of years for which the bursary was held.

Applicants may request application forms from:

MSc. Bursaries
The Computer Science Department
University of Pretoria
Hillcrest
Pretoria
0083
South Africa
Editor’s Notes

It is with sincere gratitude that SACJ takes leave of Dr Peter Lay who, until recently, was the assistant editor dealing with Information Systems. He has left academia for what sounds like a more gentle lifestyle. (He has gone farming!) Under Peter’s stewardship the number of high-quality IS papers in SACJ grew steadily. In general, IS papers tend to be accessible and relevant to a wide spectrum of computer professionals, and the quality of IS papers that have been appearing in SACJ has significantly contributed to the increased interest being shown in the journal by the local computer industry. If this growth in interest is to be sustained, it is urgent and important to find a suitable replacement assistant editor. The ideal candidate should not only be respected as an academic by his peers, but should also be disposed to enthusiastically promote SACJ in the private sector. Since a shortlist of candidates is currently being compiled, I would like issue a general appeal for names that might be included on it. Please contact me urgently if you would like to be considered for the job, or if you would like to nominate someone that you consider to be particularly suitable.

My three year term of office as editor expires in October. I have always considered it a great privilege to hold this position, and as a result, I felt honoured when the SAICS executive committee requested that I stay on for a further term. Nevertheless, I initially declined the request on the grounds that the time-demands of the job were significantly eroding my ability to fulfil other duties. Particularly demanding has been the task of seeing to the typesetting of the various contributions - either by doing it myself, or by ensuring that it is adequately done by someone else. Recently, however, Prof G de V Smit (Riel Smit) at UCT has offered to assume the role of production editor. This generous offer so much changes the complexion of what is being asked of me that I am now both willing and honoured to continue as editor for another term. I am very grateful to Riel for his offer and I look forward to working with him. In future, authors whose papers have been accepted for publication will be asked to liaise directly with him regarding the precise form in which the final contribution should be submitted.

The next issue of SACJ will consist largely of a selection of papers that were presented at the 6th South African Computer symposium. The selection will be based on comments from the referees who, at the time, were asked to adjudicate the papers in terms of their appropriateness for both the conference as well as for SACJ publication. Papers which, in the opinion of one or more referees, required major revision will have to be resubmitted to SACJ for refereeing purposes. Authors will soon be contact in this regard.

At the time of writing, the updated list of "approved" publications for the first half of 1991 had not yet been released by the relevant authorities. For the sake of past, present and future contributors I sincerely hope that SACJ will be on the list when it eventually comes out. However, I have become increasingly aware that there is a real danger of laying too much store on papers published in so-called approved journals as a basis for evaluating and rewarding research. I hope to expand more fully on this theme in a future edition of SACJ. Keep watching this space!

Derrick Kourie
Editor

This SACJ issue is sponsored by a generous donation from UNIDATA
An Investigation in the Separation of the Application from its User Interface

J H Greyling and P R Warren
Department of Computer Science, University of Port Elizabeth, P O Box 1600, Port Elizabeth, 6000

Abstract

The separation of an interactive program's application from its user interface is investigated. In order to support good human interaction, the interface requires a deep knowledge of the application. This is in direct conflict with the concept of separation of these two components. The study is concerned with seeing how well this conflict can be resolved in the context of graphics programs. Generic User Interface Package (GUIP) has been developed in Turbo Pascal 5.5, making extensive use of the inheritance and polymorphic features of object-oriented programming. Although some features considered part of the interface were not handled by GUIP, a substantial separation of the interface from the application was achieved.

Keywords: User Interface, Separation, Seeheim Model, Object Oriented Programming
Computing Review Categories: D.2.2

Received October 1990, Accepted January 1991

1 Introduction

This study arose out of work done at the University of Port Elizabeth on an interactive molecular graphics package (IMMP) [23]. It became clear that because of its poor user interface, very few of the initial users of the package would have continued using it on a regular basis. The problems with its interface had to be addressed. Unfortunately, the interface was so intertwined with the actual application that this was difficult. Separation of the user interface from the application should improve the modifiability of both these components. Just how well they can be separated while still maintaining a good interface has been studied by Szekely [20]. What this study does is to apply his notions to a conventional programming environment. This is done to see how well the concepts transform from Lisp to Pascal.

An overview of the theoretical concepts is given. This includes a discussion of the conflict between separating and maintaining a good interface, and a brief description of the Seeheim Model [8]. This article's emphasis is on the development of the Generic User Interface Package (GUIP). The way in which the Seeheim Model was interpreted, and how the conflict was resolved in our environment, is discussed.

2 Overview of the Theoretical Concepts

Since the separation of the user interface and application is being investigated, it is important to define these two components. According to Szekely [20] the application defines what the program can do, and the user interface defines how users tell the program what to do, and how the program tells the users what it did.

2.1 The Conflict

The concept of separation results from the assumption that the application and interface are independent [5]. This model is, however, in conflict with reality where the interface often requires a deep knowledge of the application. The main examples of where such knowledge is required are input, control and the display of the application data structures.

2.1.1 Input

Effective error management and feedback is extensively discussed in the literature as essential in assisting the user with input. Important issues in error management are the prevention of errors and the validation of parameters. Others include the supply of defaults, selecting from a list of valid strings and preventing the selection of meaningless operations [2, 17, 12]. Effective feedback is, for example, achieved by messages, highlighting selected objects and changing the cursor shape [17, 4]. An essential property in graphics applications is the selection of objects on the screen, referred to as picking. None of the above can be supplied by the interface without definite knowledge of the application and its data structures [20, 19, 14, 8, 18].

2.1.2 External versus Internal Control

With external control the interface invokes application modules in response to user inputs, and is thus in control. In contrast to this, with internal control, the application is in charge, and requests input from the user interface [22].
In interactive systems external control has the advantage of giving the user maximum control over the dialogue [21]. External control also supports consistency, maintainability and portability since the code dealing with user interaction is concentrated in one place [9]. It is, however, often a problem to cast a complex implementation completely in one mode [19]. The two interface properties of monitoring and aborting the execution of an operation highlight this problem. For these properties to be controlled externally by the user interface, multiprogramming and associated software interrupts would have to be implemented. The environment in which GUIP was developed does not support multiprogramming, thus execution monitoring and aborting could not be externally controlled.

2.1.3 The Application Data Structures
Ideally all output should be managed by the interface. For some applications the graphical presentation of the data structures is very complex (e.g. three-dimensional graphics). It thus proves to be impractical to design a generic interface package which manages all output. Many questions are posed in the literature on this issue [17, 9, 6, 8, 19].

The ability to undo semantic operations is an important aspect in interactive systems [14, 17, 9, 11, 8]. Since this mainly implies undoing unwanted changes to the application data structures, it is difficult for the user interface to be responsible for this feature.

2.2 The Seeheim Model
A generally accepted model for designing a user interface is the one developed at Seeheim [8], which divides the system into four modules. The Presentation Manager and Dialogue Control are the two components of the user interface (Figure 1).

- **Presentation Manager**
  The presentation manager is responsible for input and output.

- **Dialogue Control**
  This component defines the structure for the dialogue between the user and the application program.

- **Application Interface**
  The application interface serves as a buffer between the user interface and the application program.

- **Application**
  The application provides the functionality of the system in the form of a collection of semantic operations.

The rest of the paper discusses an implementation of a system in Turbo Pascal 5.5 using the Seeheim Model, to see to what extent the interface could be separated from the implementation.

3 GUIP
GUIP is a user interface management system supporting graphics applications which was developed at the Department of Computer Science of UPE. A menu driven interface consisting of a tree of items is provided. The items can be selected and activated. Items on the tree do different jobs: menu items select submenus or forms, execution items execute operations and input items request data from the user. They are collected together on forms and menus. A form is a list of input items and one or more execute items. A menu is a list of form names and submenu names.

GUIP supports many of the features required of a good interface. Sibert, Belliardi and Kamran [17] discuss most of these features. The user is able to specify a sufficient set of data types, namely positions, pick objects, quantities and strings. A variety of selection options is supported. This includes the selection of objects from the screen, items from a menu or form and strings from a list. The size and contents of lists can be determined at runtime. Immediate feedback is provided through the highlighting of selected objects, changing the cursor shape, context-sensitive messages and altering the colors of menu and form items indicating their availability, selection and activation. The simultaneous availability of input devices (e.g. mouse, keyboard and programmed function keys) is supported. Control of the dialog is mainly vested in the interface, but when necessary the application can influence the flow of control. This is supported by providing tools which can be called by the application in order to request input from the user. Error prevention is mainly achieved through preventing the selection of meaning less operations as well as invalid objects. Extensive parameter validation during the entering of data as well as before the execution of operations, supplies effective error detection. It is generally accepted that interfaces should not have unnatural ordering constraints [5, 11, 17, 1]. GUIP supports this concept by allowing the entering of data in any order and by supplying the escape mechanism.
There are however situations where sequencing is natural [5, 7]. This can be implemented by automatically sequencing the selection and activation of items on a form.

GUIP supplies tools which can be used by the application for monitoring the execution of an operation. These tools, however, need to be called from the application. The aborting of an execution as well as the undoing of unwanted changes to the application data structures, is left to the application to handle.

Turbo Pascal 5.5 was used for the development of GUIP, making extensive use of the inheritance and polymorphic features of object-oriented programming [3, 16, 5].

In GUIP the modules of the Seeheim Model are referred to as the Presentation Level, Dialogue Manager, Application Interface and Application. GUIP's interpretation of the Seeheim Model views the Presentation Level as a combination of tools responsible for input and output. The Dialogue Manager serves as an inference engine controlling the dialogue by using an application-specific menu tree. The important issue of application knowledge required by the interface is addressed in the Application Interface.

3.1 Presentation Level

The Presentation Level is viewed as a combination of tools which manages input and output, thereby hiding all physical device details from the other levels [7]. The Presentation Level supplies GKS logical devices [10] to the Dialogue Manager for the gathering of input. Item selection and activation is done using the choice device. The string device is used for entering strings, while reals and integers are entered using the valuator device. As the cursor moves on the graphics screen, the Dialogue Manager continuously requests the locator device for its coordinate position. Note that coordinates are passed via the Dialogue Manager to a pick procedure supplied by the application. This simulates a pick device. Any of the above devices can be interrupted by the activation of a hotkey. A relevant token will be returned to the Dialogue Manager.

On the output side the Presentation Level is mainly responsible for the layout of the display screen and relaying messages/prompts to the user.

3.2 Dialogue Manager

The actual control of the dialogue is situated in the Dialogue Manager. It uses the object-oriented approach, relying on the inheritance and polymorphic features of objects [3]. The menu tree is constructed in the Application Interface making use of default objects supplied by the Dialogue Manager. Each node on the menu tree is an instance of these objects, or descendants of them. The inheritance hierarchy and abbreviated definitions of the objects are given in figures 2 and 3. Note that the object TreeNode serves as an ancestor to all these objects. TreeNode has a string attribute Item, which is used by the choice device to present a node.

ParentNode has a NodeList representing nodes on the tree. Each menu on the tree will be an instance of ParentNode. Every item in the NodeList will point to a submenu or a form. FormNode is a descendant of ParentNode. A descendant of FormNode is defined for each form on the menu tree. In every descendant of FormNode, a method Execute must be redefined. The relevant procedure in the application should be called by it. Note that FormNode inherits ParentNode's NodeList, which now represents a list of form items. The following default objects have been defined for each type of item which can occur on a form: StringNode, RealNode, IntNode, XYNode, PickNode, GoNode and ExitNode.

An important feature of the Dialogue Manager is its use of the polymorphic feature of object-oriented programming [3]. Each default object has its own virtual method Activate which is called whenever the item presenting an instance of this object is activated by the choice device. A brief description of the functioning of method Activate for each object follows.

- **ParentNode**
  - Method Activate calls the choice device to display the list of items on the screen. The user can then select and activate one of them.
- **StringNode**
  - Method Activate calls the string device in the Presentation Level, requesting a string.
- **RealNode and IntNode**
  - Method Activate calls the valuator device in the Presentation Level, requesting a real or integer.
- **XYNode**
  - Method Activate continuously calls the locator device in the Presentation Level while the cursor is moved on the graphics screen.
- **PickNode**
  - Method Activate continuously calls the locator device in the Presentation Level. The coordinates received from this device are passed to the pick procedure provided by the application.
- **GoNode**
  - Method Activate calls procedure Execute, the virtual method which was redefined for the form.
- **ExitNode**
  - Method Activate exits the current level on the tree, and calls method Activate of the ParentNode which is one level higher on the tree.

Shortcuts to commonly used operations are achieved in response to hotkey tokens received from the GKS devices. The Dialogue Manager searches down the menu tree for the relevant parentnode, and then calls its corresponding method Activate.

3.3 Application Interface

In the design of GUIP, the Application Interface is viewed as the semantic support component of the Di-
logue Manager. While the Seeheim Model identifies it as a passive interface definition, GUIP views it as an active part of the Dialogue Manager [8, 5]. The Application Interface is responsible for constructing the application-specific menu tree. The important issue of supplying application knowledge to the interface, especially concerning effective error management and feedback, is also addressed here.

As mentioned above, a menu tree is constructed in the Application Interface by making use of objects defined in the Dialogue Manager. The menu tree is built up of menus, forms and form items. Each menu will be an instance of ParentNode, with its list of objects pointing to either a submenu or a form. For each form a descendant of FormNode must be defined. FormNode has a virtual method Execute which must be redefined in each of its descendants. This method is called whenever an execute item on that form is activated.

Effective error management and feedback is supported by making extensive use of the virtual methods mechanism of object-oriented programming. Virtual methods have been defined in the default objects, for generically managing properties like parameter validation, lists to select from, the cursor shape and default values for parameters. Whenever semantic knowledge needs to be supplied to forms or items, these virtual methods are redefined in descendants of the default objects.

Parameters are, for example, validated using the virtual method ParameterOK. After a parameter is entered, this function checks its validity. In the default objects ParameterOK always returns a true value, indicating a valid parameter. By redefining it in a descendant object, the semantic validity of a parameter can be checked. An invalid parameter will not be accepted, and the user will have to reenter it. The FormNode has a virtual method FormAvail, which determines whether a form can be selected or not. By redefining FormAvail the user can be prevented from selecting a meaningless operation. This same concept of redefining virtual methods is used for supplying defaults and lists of strings, highlighting selected objects and changing the cursor shape.

By using the mechanism discussed above, the Application Interface effectively becomes the semantic support component of the Dialogue Manager. The application interface gets the required semantic knowledge by calling procedures in the application itself.

### 3.4 Application

A good criterion for measuring the success of separation is the degree to which the interface is generic. GUIP has a high level of application independence. An application must, however, support some features to be compatible.

Control is vested externally in the interface. The application is viewed as a combination of functional modules, representing the different operations of the system. These modules are called by the interface in response to user actions. All data needed by the modules are gathered by the interface, and passed on to the application when an operation is activated.

A compatible application will thus consist of a set of procedures mapping onto the set of operations of the system. Data is passed via the parameters of the procedures.

The application will keep a data structure to represent a number of different objects. It is responsible for the presentation of this data structure. This is not a violation of the principle of separation. GUIP has been developed with reference to direct graphics applications. In these applications the graphical images are part of the functionality of the application [6].

In order to pick objects on the screen, the coordinates of the cursor must be related to the contents of the application data structure. It is expected of the application to provide a procedure which will assist in this. This procedure takes coordinates as input and
TreeNode = object(Node)
  Item : string;
  ...
end;

ParentNode = object(TreeNode)
  NodeList : list;
  Procedure Activate; virtual;
  ...
end;

FormNode = object(ParentNode)
  Procedure Execute; virtual;
  ...
end;

StringNode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

RealNode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

IntNode = object(RealNode)
  ...
end;

PickNode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

XYNode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

GONode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

ExitNode = object(TreeNode)
  Procedure Activate; virtual;
  ...
end;

Figure 3: Default Objects

returns a unique identifier and position of the object nearest to that position [13].

4 Experiences with GUIP

The main experience with GUIP has been with a revised version of IMMP. The initial aim with GUIP was to improve the modifiability of both the application and the interface. According to Olsen [15] an interface package must support the generation of a working prototype on the basis of basic definitions alone. It must be possible to refine and enhance the default generated interface of the prototype continuously. Finally the user interface should be extensible in the sense that new commands and capabilities can be added. GUIP supports these concepts effectively. IMMP was modified through iteration using continuous feedback from a chemistry expert. The main modifications which had to be done were the categorizing of operations on the menu tree, addition of operations, refining error management mechanisms and automating the selection and activation sequence on certain forms.

Olsen also states that a user interface package should be readily understood by both programmers and nonprogrammers [15]. With IMMP the interface could, however, not be instantiated without a considerable amount of coding. A package is currently being developed to try and automate the instantiation as far as possible. Much of the Pascal will unfortunately still have to be coded by the interface designer. It is generally expected that a generic interface tool will assist in the development of an interface without much effort. While this may be true of packages which are merely window managers, developing an intelligent interface supporting effective feedback and error management will still require substantial work.

Other experiences with GUIP include the design of a graphical circuit editor and a generic diagram editor. The package which is developed to automate the instantiation also uses GUIP. A current third year project is the development of a system for graphically designing finite state automata, using GUIP as the interface. Similar results to those with IMMP were experienced in these projects. A major advantage in the development of these systems is that the application programmers need only concentrate on the development of the respective applications.
5 Conclusion

A substantial separation of the user interface from the application has been achieved in GUIP. However, certain features sometimes considered as part of the interface are not handled. Because of the nature of direct graphics applications, the display of application data structures is left to the application to handle. The application is also responsible for the undoing of unwanted changes to the application data structures, as well as the monitoring and aborting of the execution process. There is also a substantial amount of effort required to substantiate a user interface.

The important interactive issues of feedback, error prevention and error detection are, however, effectively provided by GUIP. Using an interpretation of the Seeheim Model, it succeeds in handling the dialogue and input in an intelligent way. The conflict between separation and semantic knowledge needed by the interface, was addressed extensively by making use of the support which objects give to the development of an extensible and reusable user interface package.

Acknowledgements

The authors wish to acknowledge the CSIR for financially supporting this project, Prof J.F. de Wet for his expert evaluation of IMMP and Lester Cowley for editing this article.

References

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:

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, $\LaTeX$ 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, $\LaTeX$ 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.
Contents

GUEST CONTRIBUTION

Why all the Fuss About Neural Networks?
G Barth ................................................................. 4

RESEARCH ARTICLES

The Placement of Subprograms by an Automatic Programming System
J P du Plessis and H J Messerschmidt ........................................ 9

An Investigation into the Separation of the Application from its User Interface
J H Greyling and P R Warren ......................................................... 16

The Physical Correlates of Local Minima
L F A Wessels, E Barnard and E van Rooyen .................................... 22

An Efficient Primal Simplex Implementation for the Continuous 2-Matching Problem
T H C Smith, T W S Meyer and L Leenen ........................................... 28

Concept Network Framework for a Multi-paradigm Knowledge Base
J Kambanis ........................................................................ 32

PEST - A Microcomputer Pascal Based Expert System Shell
A G Sartori-Angus and R Neville ..................................................... 39

A Linda Solution to the Evolving Philosophers Problem
S E Hazelhurst ................................................................... 44

TECHNICAL NOTE

Knowledge Representation using Formal Grammars
S H von Solms, E M Ehlers and D J Enslin ......................................... 54

COMMUNICATIONS AND REPORTS

Book Review & Books Received ..................................................... 57
Cost-effective Visual Simulation Based on Graphics Workstations
A Caduri and D G Kourie ............................................................ 58
A Method of Controlling Quality of Application Software
T D Crossman .................................................................... 70
An Update on UNINET-ZA: The Southern African Academic and Research Network
V Shaw ....................................................................... 75