The South African Institute of Computer Scientists and Information Technologists

Proceedings

of the

1996 National Research and Development Conference

Industry meets Academia

Interaction Conference Centre, University of Natal, Durban.
26 & 27 September

Edited by
Vevek Ram
FOREWORD

This book is a collection of papers presented at the National Research and Development Conference of the Institute of Computer Scientists and Information Technologists, held on 26 & 27 September, at the Interaction Conference Centre, University of Natal, Durban. The Conference was organised by the Department of Computer Science and Information Systems of The University of Natal, Pietermaritzburg.

The papers contained herein range from serious technical research to work-in-progress reports of current research to industry and commercial practice and experience. It has been a difficult task maintaining an adequate and representative spread of interests and a high standard of scholarship at the same time. Nevertheless, the conference boasts a wide range of high quality papers. The program committee decided not only to accept papers that are publishable in their present form, but also papers which reflect this potential in order to encourage young researchers and to involve practitioners from commerce and industry.

The organisers would like to thank IBM South Africa for their generous sponsorship and all the members of the organising and program committees, and the referees for making the conference a success. The organisers are indebted to the Computer Society of South Africa (Natal Chapter) for promoting the conference among its members and also to the staff and management of the Interaction Conference Centre for their contribution to the success of the conference.

On behalf of the Organising Committee
Vevek Ram
Editor and Program Chair
Pietermaritzburg, September 1996
Organising Committee

Conference General Chairs
Mr Rob Dempster and Prof Peter Warren (UNP)

Organising Chair
Dr Don Petkov (UNP)

Secretariat
Mrs Jenny Wilson

Program Chair
Prof Vevek Ram (UNP)

Program Committee
Prof Peter Wentworth, Rhodes
Dr Milan Hajek, UDW
Prof Derek Smith, UCT
Prof Anthony Krzesinski, Stellenbosch
Dr Don Petkov, UNP
Mr Rob Dempster, UNP
Prof Peter Warren, UNP
## Table of Contents

Foreword .......................................................... i  
Organising Committee ........................................... ii  
List of Contributors .............................................. vi  

**Keynote Speaker**

*The Role of Formalism in Engineering Interactive Systems*  
M D Harrison and D J Duke ...................................... 1  

**Plenary**

*Industry-Academic-Government Cooperation to boost Technological Innovation and People Development in South Africa*  
Tjaart J Van Der Walt ........................................... 15  

*Checklist support for ISO 9001 audits of Software Quality Management Systems*  
A J Walker ................................................................... 17  

*The IS Workers, they are a-changin'*  
Derek Smith .......................................................... 29  

**Research**

*Examination Timetabling*  
E Parkinson and P R Warren .................................... 35  

*Generating Compilers from Formal Semantics*  
H Venter ..................................................................... 43  

*Efficient State-exploration*  
J. Geldenhuys .......................................................... 63  

*A Validation Model of the VMTP Transport Level Protocol*  
H.N. Roux and P.J.A. de Villiers ............................... 75  

**Intelligent Systems**

*Automated Network Management using Artificial Intelligence*  
M Watzenboeck ......................................................... 87  

*A framework for executing multiple computational intelligent programs using a computational network*  
H L Viktor and I Cloete ............................................. 89  

*A Script-Based prototype for Dynamic Deadlock Avoidance*  
C N Blewett and G J Erwin ....................................... 95  

*Parallelism: an effective Genetic Programming implementation on low-powered Mathematica workstations*  
H. Suleman and M. Hajek .......................................... 107  

*Feature Extraction Preprocessors in Neural Networks for Image Recognition*  
D Moodley and V Ram .............................................. 113
# Real-Time Systems

*The real-time control system model - an Holistic Approach to System Design*

T Considine

*Neural networks for process parameter identification and assisted controller tuning for control loops*

M McLeod and VB Bajic

*Reference Model for the Process Control Domain of Application*

N Dhevcharran, A L Steenkamp and V Ram

# Database Systems

*The Pearl Algorithm as a method to extract information out of a database*

J W Kruger

*Theory meets Practice: Using Smith’s Normalization in Complex Systems*

A van der Merwe and W Labuschagne

*A Comparison on Transaction Management Schemes in Multidatabase Systems*

K Renaud and P Kotze

# Education

*Computer-based applications for engineering education*

A C Hansen and P W L Lyne

*Software Engineering Development Methodologies applied to Computer-Aided Instruction*

R de Villiers and P Kotze

*COBIE: A Cobol Integrated Environment*

N Pillay

*The Design and Usage of a new Southern African Information Systems Textbook*

G J Erwin and C N Blewett

*Teaching a first course in Compilers with a simple Compiler Construction Toolkit*

G Ganchev

*Teaching Turing Machines: Luxury or Necessity?*

Y Velinov

# Practice and Experience

*Lessons learnt from using C++ and the Object Oriented Approach to Software Development*

R Mazhindu-Shumba

*Parallel hierarchical algorithm for identification of large-scale industrial systems*

B Jankovic and VB Bajic
Information Technology and Organizational Issues

A cultural perspective on IT/End user relationships
A C Leonard

Information Security Management: The Second Generation
R Von Solms

Project Management in Practice
M le Roux

A Case-Study of Internet Publishing
A Morris

The Role of IT in Business Process Reengineering
C Blewett, J Cansfield and L Gibson

Abstracts

On Total Systems Intervention as a Systemic Framework for the Organisation of the Model Base of a Decision Support Systems Generator
D Petkov and O Petkova

Modular Neural Networks Subroutines for Knowledge Extraction
A Vahed and I Cloete

Low-Cost Medical Records System: A Model
O A Dami and T Seipone

A Methodology for Integrating Legacy Systems with the Client/Server Environment
M Redelinghuys and A L Steenkamp

Information Systems Outsourcing and Organisational Structure
M Hart and Kavazandis

The relational organisation model
B Laauwen

The Practical Application of a New Class of Non-Linear Smoothers for Digital Image Processing
E Cloete

A Technology Reference Model for Client/Server Software Development
R C Nienaber

The Feasibility Problem in the Simplex Algorithm
T G Scott, J M Hattingh and T Steyn

Author Index
List of Contributors

Vladimir B Bajic
Centre for Engineering Research, Technikon Natal,
P O Box 953
Durban 4000

C N Blewett
Department of Accounting
University of Natal
King George V Avenue
Durban 4001

Justin Cansfield
Department of Accounting
University of Natal
King George V Avenue
Durban 4001

Tom Considine
Apron Services (Pty) Ltd
P O Johannesburg
International Airport
1600

Eric Cloete
School of Electrical Engineering
Cape Technikon
Box 652
Cape Town

I Cloete
Computer Science Department
University of Stellenbosch
Stellenbosch
7600

O A Daini
Department of Computer Science
University of Botswana
Gaborone
Botswana

Nirvani Devcharan
Umgeni Water
Box 9
Pietermaritzburg
3200

P J A de Villiers
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

Ruth de Villiers
Department of Computer Science and Information Systems
UNISA
Box 392, Pretoria, 0001

G J Erwin
Business Information Systems
University of Durban-Westville
Private Bag X54001
Durban 4000

G Ganchev
Computer Science Department
University of Botswana
PBag 0022
Gaberone, Botswana

J Geldenhuys
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

Louise Gibson
BIS, Dept Accounting & Finance
University of Durban
Pvt Bag X10
Dalbridge 4014

Mike Hart
Department of Information Systems
University of Cape Town
Rondebosch
7700

M. Hajek
Department of Computer Science
University of Durban-Westville
Pvt Bag X54001
Durban 4000

A C Hansen
Dept of Agricultural Engineering
University of Natal
Private Bag X01
Scottsville 3209

J M Hattingh
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom 2520
Boris Jankovic  
Centre for Engineering Research  
Technikon Natal  
P O Box 953,  
Durban 4000

Paula Kotze  
Department of Computer Science and  
Information Systems  
UNISA  
Box 392  
Pretoria, 0001

J W Kruger  
Vista University  
Soweto Campus  
Box 359  
Westhoven 2124

A C Leonard  
Dept of Informatics  
University of Pretoria  
Pretoria, 2000

Ben Laauwen  
Laauwen and Associates  
P O Box 13773  
Sinoville  
0129

Mari Le Roux  
Information technology, development: project leader  
Telkom IT 1015  
Box 2753  
Pretoria 0001

P W L Lyne  
Dept of Agricultural Engineering  
University of Natal  
Private Bag X01  
Scottsville 3209

Rose Mazhindu-Shumba  
Computer Science Department  
University of Zimbabwe  
Box MP167  
Harare, Zimbabwe

Meredith McLeod  
Centre for Engineering Research,  
Technikon Natal,  
P O Box 953  
Durban 4000

D Moodley  
Computer Management Systems  
Box 451  
Umhlanga Rocks  
4320

Andrew Morris  
P O Box 34200  
Rhodes Gift  
7707

R C Nienaber  
Technikon Pretoria  
Dept of Information Technology  
Private Bag X680  
Pretoria 0001

E Parkinson  
Department of Computer Science  
University of Port Elizabeth  
Box 1600  
Port Elizabeth 6000

Don Petkov  
Department of Computer Science and  
Information Systems  
University of Natal  
P Bag x01  
Scottsville 3209

Olga Petkov  
Technikon Natal  
Box 11078  
Dorpspruit 3206  
Pietermaritzburg

N Pillay  
Technikon Natal  
Box 11078  
Dorpspruit 3206  
Pietermaritzburg

Mari Le Roux  
Information technology, development: project leader  
Telkom IT 1015  
Box 2753  
Pretoria 0001
VRam
Department of Computer Science and Information Systems
University of Natal
PBag x01
Scottsville 3209

Melinda Redelinghuys
Department of Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

Karen Renaud
Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

H N Roux
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

T G Scott
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom
2520

T Seipone
Department of Computer Science
University of Botswana
Gaborone
Botswana

Derek Smith
Department of Information Systems
University of Cape Town
Rondebosch
7700

Anette L Steenkamp
Department of Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

T Steyn
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom 2520

H. Suleman
Department of Computer Science
University of Durban-Westville
Pvt Bag X54001
Durban 4000

A Vahed
Department of Computer Science
University of Western Cape
Private Bag X17
Bellville 7530

A Van der Merwe
Computer science and Information Systems
UNISA
P O Box 392
Pretoria,0001

Tjaart J Van Der Walt
Foundation for Research and Development
Box 2600
Pretoria, 0001

K Vavatzandis
Department of Information Systems
University of Cape Town
Rondebosch
7700

Y Velinov
Dept Computer Science
University of Natal
Private Bag X01
Scottsville 3209

H Venter
Department of Computer Science
University of Port Elizabeth
Box 1600
Port Elizabeth 6000

H L Viktor
Computer Science Department
University of Stellenbosch
Stellenbosch
7600

R Von Solms
Department of Information Technology
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth 6000
A J Walker
Software Engineering Applications Laboratory
Electrical Engineering
University of Witwatersrand
Johannesburg

P Warren
Computer Science Department
University of Natal
P/Bag X01
Scottsville 3209

Max Watzenboeck
University of Botswana
Private Bag 0022
Gaberone
Botswana
LESSONS LEARNT FROM USING C++ THE OBJECT-ORIENTED APPROACH TO SOFTWARE DEVELOPMENT

Dr Rose Mazhindu-Shumba
Computer Science Department
University of Zimbabwe
Box MP167 Harare
Zimbabwe: email:mshumba@zimbix.uz.zw

Abstract

This paper presents the lessons learned from using the C++ programming language and an object-oriented design methodology for the development of a prototype analysis tool for inheritance relationships in object-oriented C++ software systems. During the project two tools were developed; a data extraction tool, and an analysis tool. We noted that inheritance and dynamic binding, reuse and strong typedness are powerful features of C++. However we also noted that; C++ is an evolving language which needs standardization, the learning curve is very long, C++ leaves the programmer to resolve most problems and C++ code is very difficult to navigate and understand. We hope that the lessons explained in this paper will not only enhance the understanding of the C++ language, but also be very useful to anyone who intends using the language with any other object-oriented designing approach.

Introduction

The C++ language Stroustrup [Str86] has become one of the most popular languages for the development of object-oriented software. We feel that the popularity of C++ is due to the following facts:

C++ is derived from C, itself a popular language. Therefore C developers looking for the benefits of object-oriented programming look to C++ as a logical step towards object-oriented programming. C programmers move incrementally to C++, adding in object-oriented techniques at their own speed.

There is plenty of literature in the form of books and journals that explain and illustrate the use of the C++ language.

There is many development tools for the C++ language.

The C++ language is available on most platforms: Unix, Dos and VMS.

In this paper, we explain the lessons that we have learnt from using C++ and an object-oriented design methodology (the Rebecca Wirfs-Brock et al, [WBWW90]) in the development of an interactive computer-aided prototype analysis tool for inheritance relationships in object-oriented C++ software systems, SAOSS (A System for the Analysis of Object-Oriented Software Systems). SAOSS consists of a set of tools; a data extraction and storage tool (DEST) to extract required data from a given C++ system and store it into an object-oriented database and an analysis tool to analyse the data in the database. SAOSS was implemented on the Unix platform using the Sun 2.1 C++ compiler, the OSE C++ third party libraries by Dumpleton, [Dum93], and a graphical interface development toolkit, tcl-tk by Ousterhout [Ous93]. In all a one hundred classes were designed and implemented.

We hope that the lessons explained in this paper will not only enhance the understanding of the C++ language, but also be very useful to anyone who intends using the language with any other object-oriented designing methodology.
Lessons learnt from using C++

Inheritance and dynamic binding a powerful feature

From the development of SAOSS, we found that polymorphism, inheritance and dynamic binding together form a powerful feature of C++. As part of (DEST), we have some classes as presented below:

- An abstract class DETAILS with a pure virtual function get-details().
- Classes INHERITANCE-DATA and METHOD-DATA publicly inherit
- Client C++ code given below the classes.

In C++, a pointer to a base can point at either the base class object or a derived class object. The member function selected depends on the classes of the object being pointed at. The implication of the statement: `p[i] -> get-details()` is the following:

Invoke at run time the get-details() member function, which corresponds to the derived type of the object that `p[i]` is pointing to. If the object's class has no override for get-details(), use the parent class get-details().

The power of inheritance and dynamic binding

```cpp
class DETAILS
{
public:
    virtual double get-details() = 0;
};

class INHERITANCE-DATA: public DETAILS
{
    double height, width;
    public
    double get-details();
};

class METHOD-DATA: public DETAILS
{
    double radius;
    public
    double get-details();
};

client C++ code
DETAILS *p[N];
for (i=0;i<=N;++i)
    get-details=p[i]->get-details()
```

When the client code is executed, the selection of which method to invoke for a virtual function is dynamic. The advantage of such a design that uses inheritance and dynamic binding is that if more classes that represents data to be extracted from given system are added to the inheritance hierarchy, our code needs not change. This helps create open-ended systems.
Reuse support in C++ is a powerful feature

In C++, code reuse is achieved by use of aggregation, templates and private inheritance. We found aggregation and templates very useful in the development of SAOOSS.

Aggregation

Aggregation is the containment of an object of a class inside another class. There were several cases in the development of the analysis tool where we only needed to reuse a small percentage of the parent methods. In such a case it was less appropriate to use public inheritance since the is-a-kind of relationship did not exist among the classes. Since what we wanted was reuse of some functionality (not all), aggregation was used. Although we could have used private inheritance, we felt aggregation was better. Aggregation preserves the encapsulation of the contained objects, whereas when private inheritance is used, the inheriting client can have access to protected members of the base class. The fact that aggregation preserves encapsulation means that it reduces coupling among classes. Two objects are coupled if and only if at lease of them acts upon the other. Any evidence of a method of one object using methods or instance variables of another object constitutes coupling [CK94].

Templates or parametric polymorphism

We found templates to be very useful in the development of SAOOSS. all the classes in the OSE C++ data structure library are templated. Whenever we wanted a specific type of data in the structures, all we needed doing was instantiate the type. This saved us a lot of development time and effort and helped produce consistency across the developed system. However, the current problem with templates is that most available compilers do not support them as yet, since they are a new feature recently introduced into C++.

Strong typedness is a powerful feature of C++

C++ is a strongly typed language. A strongly typed language is one in which it can be guaranteed that all expressions have correct types at compile time. The strongly typedness of C++ is due to the fact that it has many fundamental types, it lets programmers define new types and has more restrictions on the operations allowed on the instances of the type. C++ checks the message names and types of arguments in a class function call at compilation. The message must be defined for the object class. In addition all the arguments must be types derived from the types declared for the functions arguments.

The importance of checking types at compile time is that in object-oriented programming, member functions can be misused. The misuse of a message may not be obvious when a program's source is checked. In addition since inheritance distributes the functions defined for a class, it may not be obvious whether a function is available to a particular class.

The C++ learning curve

Learning the C++ language is not very straight forward, the syntax of the language is not very consistent, especially when coming from a Pascal background. The learning of the syntax often distracts one from more important issues like the use of the object-oriented features.

Once the syntax is mastered the next step is to learn how to use the object-oriented features of C++. Learning to use the object features of C++ requires climbing a long learning curve. It takes several months to really appreciate the power of C++ classes and to become class designers. In fact it takes much more time to come to terms with the technical features of object-oriented programming than to move to another procedural language. We feel that it is very difficult to use the object-oriented features of C++ because C++ is a general purpose language that supports object-oriented programming among other things. In addition C++ class constructs come with a variety of protection mechanism, virtual function, virtual base classes, const member functions, reference variables. These features place a heavy burden on programmers trying to move to C++.
We feel that learning C++ and object-oriented programming is an ongoing process which will never end since C++ is continuously evolving. C++ is an evolving language, which needs standardization.

From our experiences of using C++ and reading other people's experiences, we feel that there is a very urgent need for the standardization of the C++ language. No doubt C++ is perceived as one of the languages that is best suited for modern software development. At the moment there is great variety of C++ implementations. Many of the available implementation produce their own interpretation of some C++ features. We feel that in the absence of real standards for language definition, future implementations will inevitably lead to multiple dialects. There is great need for established conventions and standards for design documents and coding.

Many development projects that find C++ to be the most appropriate language for their needs do not choose to utilise it because of the instability of the language definition, Lenkov [Len92]. C++ is a language that is still evolving and changing too rapidly to be a safe choice for large software development.

Since we started working with C++ in 1990, several new features were added to the language by Standard Committee. These include templates, exception handling and rules for relaxing the return types for virtual functions.

**C++ leaves the programmer to resolve most problems**

The C++ language assumes programmers know what they are doing and the available language constructs should make it easy to do what is desired.

Programmers have to figure out improper overriding of member functions

Whenever a virtual function is declared in C++, the overriding function must have the same signature as the virtual function in the ancestor. If the overridden signature does not match the signature of the virtual member function in the ancestor class, C++ treats the member functions as two distinct member functions with no relationship to each other. C++ leaves the programmer to figure out where a member function is not properly overridden. We feel that C++ should at least warn us or report this as an error. The consequences of leaving such a resolution to the programmer, can result in unpredictable results.

**Mistakes in overriding a member function**

```cpp
class NUMBER
{
public:
    virtual void add(int x)
    {
        return x*x;
    }
};

void main()
{
    NUMBER *p = new BIGNUMBER;
    cout << p->add(3) << endl;
}
```

Support we have a parent class NUMBER, child class BIGNUMBER and a main function as shown above. The reader of the program may expect 30 as the printed answer. We can justify this in the
following way, 'add' is virtual function NUMBER. In function main, a pointer to a
instance of class BIGNUMBER is substituted for the pointer to class NUMBER. The function 'add(3)' is then invoked
through the object pointer p. We would expect the add function in the child class to be invoked to
produce an output 30. However the redefined function 'add' in BIGNUMBER violates the requirements
for overriding a member function. The signature of the function does not match the signature of the
member function in the parent class. Therefore static binding is used and the version of 'add' in parent
is bound to the object of type child, producing an output of 9.

Programmers have to choose which member functions to make virtual and which to make ordinary

C++ leaves the programmer to choose which functions are to be dynamically bound and which functions
are to be statically bound. How can a designer of a class hierarchy, for example in a class library know
in advance which functions some clients intend to require dynamic binding? Some researchers have
proposed that it might be wise to declare all C++ functions as virtual Coplien [Cop92], Weiner [Wei94]

Programmers have to choose which ancestor classes to make virtual

C++ resolves the repeated inheritance problem by using virtual base classes. A designer of a class
library is again forced to make decisions about which classes to make virtual. Once a class is make
virtual, it cannot be undone by a client who wishes to create a class which has two copies of the ancestor.
The class hierarchy designer actually imposes decision about the sharing in the descendant class.

Programmers have to resolve member function and data member name conflicts.

Whenever multiple inheritance is used, it is possible that there maybe member functions and data member
name clashes from multiple ancestors. C++ leaves the programmer to resolve such clashes. The user
has got to add code into the child class to resolve the clashes.

C++ emphasizes user defined memory management

A memory-safe language is one which a user's program cannot unknowingly corrupt the contents of
memory Reed and Wyant [RW92]. C++ is defined in such a way as to prevent the compiler from
statically checking memory operations. One of the most undetected programming errors is the index
outside an array bound. If the index is large enough the operating system may signal a fault else the
erroneous index goes undetected. The value may later be passed into other parts of the program and
some data structures maybe corrupted. In addition, C++ objects can be created using the new operator.
The programmer has to figure out when it is safe to deallocate an object. The only time it is safe to
deallocate an object is when there are no more references to it. As soon as an object is deallocated the
C++ language runtime routine makes the object available for a subsequent instance allocations. We end
up with situations where two different sections of a program are using the same piece of storage to
represent two entirely different objects. This causes program crashes. The way C++ handles memory
management is in contrast with other object-oriented languages such as Smalltalk and CLOS. In
Smalltalk and CLOS, unused objects are automatically reclaimed by the system. The programmer
is then free to worry about more substantial issues.

The fact that C++ leaves most problems to the programmer may seem a good idea. A programmer is
left with the maximum flexibility of expression. From our experience, this is bad in that programmers
are too often tempted to select poor tradeoffs because they do not understand the multitude of options
available.

It is very difficult to navigate and understand C++ code

C++ is an operator intensive language. Many operators maybe used with different meanings in different
context. For example the operator '=' can be used to mean initialise, assign or establish a pure virtual
function. This can lead to very difficult-to-read code. Another example which shows the difficulty of
navigating and understanding C++ code is where we have a member function 'get-name()' attached to
object M. M -> get-name()

Given such a statement on its own, it is very difficult to figure out which class the method 'get-name()' is attached. The possibilities are that:

'get-name()' is a member function of class EMPLOYEE of which M is an instance.
'get-name()' is a member function of one of the ancestor classes of the EMPLOYEE class, if the subclass has not overridden the 'get-name()' method.
'get-name()' is a member function attached to subclass of EMPLOYEE class, M is a pointer to an EMPLOYEE object and 'get-name' function is virtual.

The importance of navigating and understanding code for maintenance purposes can not be overemphasized. The fact that navigation through C++ source code is a serious impediment to maintenance has also been conformed by Wiener [Wie94], Wybolt [Wyb90].

C++ programming environments

A programming environment is an integrated set of interactive tools that include at least a language interpreter, an editor, code browse, debugger and code finding systems. C++ offers no standard class libraries except the iostream library. There are plenty of third party vendors that are now offering special purpose C++ libraries. The areas of application include graphical user interface, mathematics and data structure libraries.

Conclusion

In this paper we have presented our experiences from using the C++ programming language and an object-oriented design methodology for the development of a prototype analysis tool for inheritance relationships.

We noted that inheritance, polymorphism and dynamic binding, reuse and strong typedness are very powerful features of the C++ language. However we also noted that C++ is a language with the following properties:

- a long learning curve,
- very difficult to understand,
- leaves the programmer to resolve most problems, and
- needs standardization.

The fact that C++ leaves most of the decisions to the programmer and the fact that the C++ code is very difficult to understand suggest that there is need for more comprehensive tools to help developers understand the design decisions in developed systems. We hope that the lessons explained in this paper will not only enhance the understanding of the language, but also be very useful to anyone who intends using the language with any other object-oriented design methodology.

References


