MANOEL THEODORO FAGUNDES CUNHA

A PRACTICAL APPROACH IN THE DEVELOPMENT OF ENGINEERING APPLICATIONS TO THE INTERNET USING OBJECT ORIENTED PROGRAMMING

Dissertação apresentada como requisito parcial á obtenção do grau de Mestre em Ciências. Programa de Pós-Graduação em Métodos Numéricos em Engenharia. Setor de Tecnologia Universidade Federal do Paraná.

Orientadores : Prof. Dr. Sérgio Scheer
Prof. Dr. Waldyr de Lima e Silva Jr.

CURITIBA
1999

MANOEL THEODORO FAGUNDES CUNHA

A PRACTICAL APPROACH IN THE DEVELOPMENT OF ENGINEERING APPLICATIONS TO THE INTERNET USING OBJECT ORIENTED PROGRAMMING

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciências. Programa de Pós-Graduação em Métodos Numéricos em Engenharia. Setor de Tecnologia, Universidade Federal do Paraná

Orientadores : Prof. Dr. Sérgio Scheer

Prof. Dr. Waldyr de Lima e Silva Jr.

CURITIBA

1999

MANOEL THEODORO FAGUNDES CUNHA

A PRACTICAL APPROACH IN THE DEVELOPMENT OF ENGINEERING APPLICATIONS TO THE INTERNET USING OBJECT ORIENTED PROGRAMMING

Dissertation presented as partial fulfillment for obtaining the degree of Master of Science. Post Graduate Programme in Numerical Methods in Engineering. Sector of Technology, Federal University of Paraná, Brazil.

Advisors : Prof. Dr. Sérgio Scheer

Prof. Dr. Waldyr de Lima e Silva Jr.

CURITIBA

1999

MANOEL THEODORO FAGUNDES CUNHA

A PRACTICAL APPROACH IN THE DEVELOPMENT OF ENGINEERING APPLICATIONS TO THE INTERNET USING OBJECT ORIENTED PROGRAMMING

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre em Ciências no Curso de Pós Graduação em Métodos Numéricos em Engenharia da Universidade Federal do Paraná pela comissão formada pelos professores :

Orientador : Prof. Dr. Sérgio Scheer [illegible]

Setor de Tecnologia, UFPR

Orientador : Prof. Dr. Waldyr de Lima e Silva Jr. [illegible]

Setor de Tecnologia, UFPR

Prof. Dr. Bruno Feijó [illegible]

Departamento de Informática, PUC-Rio

Prof. Dr. Ricardo Mendes Jr. [illegible]

Setor de Tecnologia, UFPR

Curitiba, 4 de outubro de 1999

Ultimately, it comes down to taste. It comes down to expose yourself to the best things that humans have done and then try to bring those things to what you do.

Steve Jobs

Essencialmente, trata-se de experimentar. Trata-se de expor a si mesmo às melhores coisas que os homens fizeram e tentar trazer estas coisas para o que você está fazendo.

Steve Jobs

To my daughters Carolina and Cecilia,

my love and maybe a bearing

ACKNOWLEDGEMENTS

To the family for the support.

To the teachers for the incentive.

To the friends for the joy.

CONTENTS

LIST OF FIGURES

RESUMO

Este trabalho é um tutorial prático para apresentar os conceitos básicos da Programação Orientada a Objetos e o seu uso no desenvolvimento de aplicações de Engenharia para a Internet. Durante décadas, técnicas e ferramentas de programação têm mudado para acompanhar a evolução do *hardware*. O avanço do conhecimento humano também traz complexidade para as aplicações e exige o mesmo avanço nos recursos e técnicas de *software*. Entretanto, muitos estudantes e pesquisadores envolvidos com programação de computadores ainda utilizam ferramentas e técnicas muito antigos. Ainda que estes programadores estejam tratando de teorias matemáticas e de engenharia muito avançadas, boa parte deste avanço é perdido pelo o uso de técnicas de programação desenvolvidas décadas atrás. A simplicidade obtida com o uso de uma abordagem tradicional já não é mais compatível com a complexidade dos problemas em estudo. A solução de problemas de engenharia é o objetivo primário do estudante ou pesquisador. Entretanto, depois da solução do problema, o programador deve enfrentar a tarefa de criar uma interface para o usuário. Este trabalho também aborda o problema de dar à aplicação uma interface de usuário padrão sem o conhecimento profundo de programação. A Internet é o ambiente proposto neste trabalho como uma opção para o programador de engenharia. O último benefício da abordagem proposta aqui é a portabilidade de plataforma, não apenas para o código mas também para o programador.

ABSTRACT

This work is a practical tutorial to present the basic concepts of Object Oriented Programming and its use in the development of engineering applications to the Internet. During decades, programming tools and techniques have changed or created to follow hardware evolution. The development of human knowledge also brings complexity to applications and demands the same increase in software resources and techniques. However, many students and researches involved with computer programming still use very old tools and techniques. Despite those programmers are dealing with advanced mathematical and engineering theories, a great deal of their advance is lost by using programming techniques developed decades ago. The simplicity obtained when using a traditional approach is no longer compatible with the complexity of the problems in study. The solution of engineering problems is the primary goal of the student or researcher. However, after the solution of the problem, the programmer must face the job of creating a user interface. This work also addresses the problem of giving to scientific and engineering applications a standard user interface without deep programming knowledge. Internet is the environment proposed in this work as an option to the scientific and engineering programmer. The last benefit of the approach proposed here is the platform portability, not only for the code but also for the programmer.

1. INTRODUCTION

1.1 MOTIVATION

Over the last decades, since the invention of the first computers, programming techniques have been changed to follow hardware evolution. Computers with high processing capabilities and equipped with accessories like scanners, CD-ROM writers and laser printers can be found in supermarkets and are intended for office and home usage. High-resolution graphics boards and monitors are used to produce special effects, many of them seen in the movie industry. Users can interact with computers with a simple mouse or even with voice commands.

However, all hardware improvements are closely followed by software evolution. *Event-driven programming* is one of the new paradigms created to accommodate the hardware evolution. Many common tasks performed by a user today simply could not be implemented using old programming techniques. New programming approaches allow the use of simultaneous input devices, like a keyboard and a mouse, and the user - not the programmer - decides how to use the computer.

The hardware improvement is not the only reason for the software evolution. While the first computers were created for scientific purposes, like ballistic path calculations, the new computers are used in simulations like nuclear reactions, DNA chains and genetic research, among others. Many mathematical theories developed in the last century can be implemented, tested and proved only with the use of computers. The development of human knowledge also brings complexity to applications and demands the same increase in software resources and techniques.

However, many students and researches involved with computer programming still use very old tools and techniques. Despite those programmers are dealing with advanced mathematical and engineering issues, a great deal of their advance are lost by using of a computer technique developed decades ago. The simplicity obtained when using a half-dozen instructions are no longer compatible with the complexity of the problems in study. Even newer languages, created to teach programming, are not fitted to numeric-intensive programming, image processing, scientific visualization and real-world applications.

This work is addressed to graduate and post-graduate students who want to learn *Object Oriented Programming*. This programming technique is a well-established paradigm of the software engineering discipline and founded in many *Application Programming Interfaces (API)*. Java, one of the most recent programming languages, has been created under this new approach. Some of new compiling environments commonly used today create object-oriented code. *Object Oriented Programming* aims the development of complex applications and allows the user to write code more efficiently, going along with hardware evolution.

The solution of engineering problems is the primary goal of the student or researcher. However, after the solution of the problem, the programmer must face with the job of creating a user interface. *Graphic User Interfaces (GUI)* like Windows provide standards for data input and output operations. Well-known widgets like push button, check boxes, radio buttons, edit boxes, scroll bars, icons, among others, are used in a daily basis and become familiar to all users, without regard to their level of expertise. However, the Windows environment has about 2500 functions to deal with its graphic interface and other features. Network capabilities is another example of feature embedded in many recent applications that demand additional knowledge from the programmer.

This work also addresses the problem of giving to the engineering application a standard user interface without deep programming experience or knowledge of large libraries. Another goal of this work is to provide the programmer with tools to develop networked applications. The last benefit of the approach proposed here is the platform portability, not only for the code but also for the programmer. Internet is the environment proposed in this work as an option to the scientific and engineering programmer.

The Internet has become an integral part of the computing community. From its beginning as *ARPANET*[16], it has grown into a global means of communicating and conducting business. Currently, the Internet is projected to be central to the future of both academic and business computing. Many people, even the most occasional of the computer users, already have electronic mail (e-mail) and many World Wide Web users who have not got a home page yet are trying to learn how to build one. The current growth of the Internet - and in particular the Word Wide Web - is influencing everything in the computing industry. A significant number of software and hardware companies are working to build Internet capabilities into their current offerings.

Combining the two topics, *Object Oriented Programming* and *Internet*, a programmer can develop applications of any level of complexity and give it a standard interface. The Internet also allows easy access to the application by the academic community. Internet allows easy dissemination of research results as well as later contributions to the improvement of the applications.

1.2 HISTORICAL ASPECTS

Object Oriented Programming[13,17,26] is a very well known programming technique and has been studied by a large number of students and researchers over the last years. Many scientific and engineering works have been written on this theme. However, in previous works on this subject, the engineering authors have focused on the solution of specific problems, mainly in the fields of Computer Graphics[7,8], Finite Elements[30,31] and Boundary Elements Methods. In most of those works, the main concern was the creation of a class hierarchy that addressed the data structures of the applications. This work approaches the implementation of the program itself using Object Oriented Programming.

The dissemination of the Internet in the academic, business and personal environments attracts many students in the investigation of its potential. Many educational applications have been developed and used by much the academic community around the world. Daily, countless pages are made available to teach a large range of disciplines using resources like Java, Virtual Reality Modeling, among others. Engineering commercial applications, however, have not taken advantage of Internet capabilities yet. The actual versions of those programs are still running only as desktop or local network applications.

The best classical and practical references to Object Oriented Programming are books, available through Internet bookshops. The sample codes reviewed in the present work are originally presented as Fortran codes by Brebbia and Dominguez in **Boundary Elements, An Introductory Course**[1]. This book is the foundation upon which this work was built.

1.3 ORGANIZATION

This work has two main subjects : Object Oriented Programming and Internet. Each topic is also presented in two parts : theory and practice. The theory of each issue are introduced and then applied with engineering examples. The second chapter presents the basic concepts of Object Oriented Programming. The introduction of each new feature is followed by an example using the new approach and compared with the same code, written in the traditional technique. The presented theory is applied in Chapter 3 to review some computer codes, written in a traditional programming language with a very conservative technique, and to rewrite them using the of Object Oriented Programming approach. The reviewed programs were presented by Brebbia and Dominguez[1] to solve potential and elastostatics problem using the Boundary Elements Method. In Chapter 4, Internet essentials are introduced and placed in perspective. Each development tool is considered under the objective of giving to the applications client-side or server-side capabilities. Chapter 5 combines the two paradigms, Object Oriented Programming and Internet in order to give to the applications presented in Chapter 3 a user interface easy to develop. The last chapter contains the conclusions and recommendations. Limitations of each programming method are discussed and directions for future research are suggested.

2. OBJECT ORIENTED PROGRAMMING CONCEPTS

Object Oriented Programming[20] is a programming technique created to allow the development of large and complex programs.

Since the invention of the first computers, approaches to programming have changed in order to accommodate the increasing complexity of the programs. First programs were just a few hundred binary instructions. As programs grew, assembly language was invented to deal with larger, increasingly complex programs using symbolic representations of the machine instructions. High-level languages as Fortran have been introduced to give the programmer better tools to handle complexity. With the appearance of structured languages, like Pascal and C, it was possible to write moderately complex programs more easily. However, even using structured programming methods, once a project reaches a certain size, its complexity becomes too difficulty for a programmer to handle.

Object Oriented Programming takes the best features of structured programming, combines them with new concepts, and allows to easily decomposing a problem into subgroups of related parts. Then, one can translate these subgroups into self-contained units called *objects*.

All Object Oriented Programming languages have three features in common : *objects*, *polymorphism* and *inheritance*.

2.1 OBJECTS

The single most important feature of an object-oriented language is the *object*. In a simple manner, an *object* is a logical entity containing variables and functions that manipulate those variables.

Within an *object*, some of the variables or functions may be *private* to the *object* and inaccessible by the rest of the program. In this way, an *object* provides a significant level of protection against modification or incorrect use. This linkage of code and data is often referred to as *encapsulation*.

Consider a simple C program :

```
#include <iostream.h>
void main(void) {
  int n;
  n = 10;
  cout << n << '\n';
}
```

In this code, an integer variable is created and its content is defined and displayed directly with simple assignment and output statements.

The object-oriented version of the same program would be :

```
#include <iostream.h>
class number {
int n;
public:
  void setnum(int a) { n = a; }
  int getnum(void) { return n; }
};
void main(void) {
  number obj;
  obj.setnum(10);
  cout << obj.getnum() << '\n';
}
```

The new code begins with a block that declares an *object* with one *private* variable, *n*, and two *public* functions, *setnum* and *getnum*. In the program body, the *object* is created and the variable is accessed through the functions. The variable is *private* and cannot be accessed directly by the program. The functions of the *object* are used to define and display the content of that variable.

2.1.1 CLASSES

To create an *object* one needs first to define its general form using the keyword *class* :

```
class classname {
  //private variables
public:
  //public functions
};
```

A *class* can contain *private* as well as *public* members. By default, all members defined in the *class* are *private*. *Private* variables cannot be accessed by any function that is not a member of the *class*. One can also define *private* functions, which can only be called by other functions of the *class*.

To make parts of a *class* accessible to other parts of the program, one must declare them after the keyword *public*. All variables and functions defined after *public* are accessible by all other functions in the program. Generally, the rest of the program accesses an object through its *public* functions. Although one can have *public* variables, their use should be avoided or eliminated. Instead, one should make all data *private* and control access to it and through *public* functions. This will help preserve *encapsulation*.

To code a function that is member of a *class*, one must inform the compiler to which *class* the functions belong using the *scope resolution operator (::)*.

```
class number {
  int n;
public:
  void setnum(int);
  int getnum(void);
};
void number::setnum(int a) {
  n = a;
}
int number::getnum(void) {
  return n;
}
```

Several different *classes* can use the same function names. The compiler knows which function belong to which *class* because of the *scope resolution operator* and the *class* name.

Simple *class* functions can also be coded within the *class* declaration :

```
class number {
  int n;
public:
  void setnum(int a) { n = a; }
  int getnum(void) { return n; }
};
```

Once the *class* is declared, one can create an *object* in the same way a variable is created but using the *class* name instead of the variable type. For all intents and purposes, an *object* is a variable of an user-defined type.

```
#include <iostream.h>
class number {
  int n;
public:
  void setnum(int a) { n = a; }
  int getnum(void) { return n; }
};
void main(void) {
  number obj;
  obj.setnum(10);
  cout << obj.getnum() << '\n';
}
```

To call a member function from a part of the program that is not part of the *class* one must use the *object* name and the *dot operator*. Only when a member function is called by code that does not belong to the *class* the *object* name and the dot operator must be used. Otherwise, one member function can call another member function directly, without using the *dot operator*.

2.1.2 CONSTRUCTORS AND DESTRUCTORS

As it is usual to initialize a variable at the time it is declared, it is very common for some part of an *object* to require initialization before it can be used. Because the requirement for initialization is so common, Object Oriented Programming allows *objects* to initialize themselves when they are created.

This automatic initialization is performed with a *constructor* function. A *constructor* function is a special function that is a member of the *class* and has the same name of that *class*.

Consider the simple program :

```
#include <iostream.h>
void main(void) {
  int n = 10;
  cout << n << '\n';
}
```

In the code above, the integer variable *n* is created and its content is defined at the time it is declared.

The object-oriented version of the same program would be :

```
#include <iostream.h>
class number {
  int n;
public:
  number(void) { n = 10; }
  int getnum(void) { return n; }
};
void main(void) {
  number obj;
  cout << obj.getnum() << '\n';
}
```

The new code declares the *class number* with a *constructor* function that initializes the content of the *private* variable when the *object* is created. The *constructor* function has the same name of the *class*, *number*, and it is called when time the *object* is created.

The complement of the *constructor* is the *destructor*. In many circumstances, an *object* needs to perform some action or actions when it is destroyed. There are many reasons why a *destructor* function may be needed; for example, to set free memory previously allocated. The *destructor* has the same name as the *constructor* but is preceded by a tilde (~).

2.2 POLYMORPHISM

Object Oriented Programming languages support *polymorphism*, which allows a name to be used for several related but slightly different purposes. The purpose of *polymorphism* is to let a name to specify a general class of action. Depending upon what type of data it is dealing with, a specific instance of the general case is executed.

The way Object Oriented Programming achieves *polymorphism* is trough the function *overloading*. In Object Oriented Programming two or more functions can share the same name as long as their parameter declarations are different. In this situation, the functions sharing the same name are said to be *overloaded*. The main advantage of using *overloaded* functions it that they allow related sets of functions to be accessed using a common name. In a sense, function *overloading* lets one create a generic name for an operation. The compiler knows which function to use in each case because the type of the argument.

For example, consider the sample code :

```
#include <iostream.h>
class rectangle {
  double area;
public:
  void setarea(double x) { area = x; }
  void setarea(double a, double b) { area = a * b; }
  double getarea(void) { return area; }
};
void main(void) {
  rectangle obj;
  obj.setarea(12);
  cout << obj.getarea() << '\n';
  obj.setarea(3,4);
  cout << obj.getarea() << '\n';
}
```

In the above code, the value of *area* is defined by the function *setarea*. By using *polymorphism,* the *class* can be improved with a new function to define the value of area. Notice the use of two related functions with the same name, *setarea*. The compiler selects the correct function depending on the parameters with which it is called.

2.3 INHERITANCE

Inheritance is the process by which an *object* can acquire the properties of another *object*. Without the use of *inheritance*, each *object* would have to define all of its characteristics explicitly. Using *inheritance*, an *object* needs only to define those qualities that make it unique within its class.

In Object Oriented Programming, *inheritance* is supported by allowing a *class* to incorporate another *class* in its declaration :

```
class parent {
  //private variables
protected :
  // protected members
public:
  //public functions
};
```

```
class child : public parent {
  //private variables
public:
  //public functions
};
```

A *class* that is *inherited* by another *class* is called the *base class*. The *class* that does the *inheriting* is called the *derived class*.

It must be noticed that a *public* member of a *class* can be accessed by any other function in the program. A *private* element can be accessed only by member functions.

When a *class inherits* another *class*, all *private* elements of the *base class* are inaccessible to the *derived class*. The elements of the derived *class* can access *public* functions of the *parent* but cannot access its *private* variables.

A *base class* can grant access of its *private* elements to a *derived class* by making them *protected*. The *protected* members of the class can be accessed by a *derived class* although they are still inaccessible to the rest of the program. Making an element *protected* restricts its access only to the member functions of the *class* but allows this access to be *inherited*.

When an element is *private*, access is not *inherited*. All *protected* and *public* elements of the *base class* become *protected* and *public* elements of the *derived class*, respectively.

2.4 ADVANCED FEATURES OF OBJECT ORIENTED PROGRAMMING

The basic concepts presented up to now are enough to allow the programmer without great computing skills to develop object-oriented applications. However, Object Oriented Programming is featured with advanced resources, commonly used by more experienced programmers. Those advanced features are not used in this work but need to be presented to give an overview of how much complex applications can be developed.

The most important of these features are :

- Parameterized *constructors* : Often when an *object* is created it is necessary, or desirable, to initialize various data elements with specific values. Using *parameterized constructor* functions it is possible to initialize *objects* using values known only when the *object* is created.
- Friend functions : It is possible for a nonmember function of a *class* to have access to the *private* parts of that class by declaring it as a *friend* function of the *class*.
- Multiple *inheritance* : It is allowed to a *class* to *inherit* attributes from more than one *class*.
- Run-time *polymorphism* : Pointers to objects are similar to pointers to variables. A *base class* pointer may point to an *object* of a *class derived* from that *base*. A *virtual* function is a function that is declared as *virtual* in a *base class* and redefined in one or more *derived classes*. *Virtual* functions are special because when one of them is accessed using a pointer, the compiler determines which function to call at run-time based on the type of *object* pointed to. Because of the restrictions and differences between *overloading* normal functions and *overloading* virtual functions, the term *overriding* is used to describe virtual function redefinition.
- *Pure virtual* functions and *abstract* types : When a *virtual* function that is not *overridden* in a *derived class* is called from an object of that derived class, the version as defined in the base class is used. A *pure virtual* function is a function declared in a *base class* that has no definition to the *base*. Any *derived class* must define its own implementation of the function. If a *class* has at least one *pure virtual* function, that *class* is said to be *abstract*. *Abstract classes* have an important feature : they cannot be used to declare an *object*. Instead, an *abstract class* must be used only as a base that other classes will inherit.

Many other advanced features are available to the experienced programmer, but a complete list of them is beyond the scope of this work.

3. OBJECT ORIENTED PROGRAMMING IMPLEMENTATION

In this chapter, the basic concepts of *classes* and *constructors* will be applied to review some computer codes written in a traditional programming language with a very conservative technique and rewrite them using the Object Oriented Programming approach. The programs reviewed here are presented by Brebbia and Dominguez[1] and solve potential and elastostatics problems using the Boundary Elements Method :

POCONBE	: Potential Problems using Constant Elements
POLINBE	: Potential Problems using Linear Elements
ELCONBE	: Elastostatics Code using Constant Elements

Each program defines some global variables and the functions shown below :

POCONBE	**POLINBE**	**ELCONBE**
INPUTPC	INPUTPL	INPUTEC
GHMATPC	GHMATPL	GHMATEC
EXTINPC	EXTINPL	EXTINEC
LOCINPC	LOCINPL	LOCINEC
SLNPD	SLNPD	SLNPD
INTERPC	INTERPL	INTEREC
OUTPUTPC	OUTPUTPL	OUTPUTEC
		SIGMAEC

Despite their different names, all the functions at the same row in the table above have the same purpose :

INPUTxx	: reads all the input data required by each program
GHMATxx	: computes the G and H matrices
EXTINxx	: called by GHMATxx and INTERxx
LOCINxx	: called by GHMATxx
SLNPD	: solution of linear systems of equations
INTERxx	: performs some calculations at internai points
SIGMAxx	: called by INTERxx in elastostatics problems

By using Object Oriented Programming techniques one can create a *class* to each program, where global variables are *encapsulated* as *private* members and the functions with equal purposes can be declared with the same name, as shown in the samples ahead.

3.1 POTENTIAL PROBLEMS USING CONSTANT ELEMENTS

The first code reviewed is a simple computer code for solving Laplace type problems. The main program defines general integer variables, an integer one-dimensional array and some real arrays to store data and results, as shown below :

```
  PROGRAM POCONBE
  DIMENSION X(101),Y(101),XM(100),YM(100),FI(100),DFI(100)
  DIMENSION KODE(100),CX(20),CY(20),POT(20),FLUX1(20),FLUX(2)
  COMMON/MATG/ G(100,100)
  COMMON/MATH/ H(100,100)
  COMMON       N,L
C READ DATA
  CALL INPUTPC(CX,CY,X,Y,KODE,FI)
C COMPUTE THE H AND G MATRICES AND FORM SYSTEM ( A X = F )
  CALL GHMATPC(G,DFI,D,100)
C SOLVE SYSTEM OF EQUATIONS
  CALL SLNPD(G,DFI,D,100)
C COMPUTE THE POTENTIAL AND FLUXES AT INTERNAL POINTS
  CALL INTERPC(FI,DFI,KODE,CX,CY,X,Y,POT,FLUX1,FLUX2)
C PRINT RESULTS AT BOUNDARY NODES AND INTERNAL POINTS
  CALL OUTPUTPC(XM,YM,FI,DFI,CX,CY,POT,FLUX1,FLUX2)
  STOP
  END
```

In order to compute the coefficients of G and H matrices the GHMATPC subroutine calls two additional subroutines, EXTINPC and LOCINPC, as shown in the fragment of code below :

```
   SUBROUTINE GHMATPC(X,Y,XM,YM,G,H,FI,DFI,KODE)
   ...
C  COMPUTE THE COEFFICIENTS OF G AND H MATRICES
   DO 30 I=1,N
   DO 30 J=1,N
   KK=J+1
   IF(I-J)20,25,20
20 CALL EXTINPC(XM(I),YM(I),X(J),Y(J),X(KK),Y(KK),
  1G(I,J),H(I,J),DQ1,DQ2,DU1,DU2,0)
   GOTO 30
25 CALL LOCINPC(X(J),Y(J),X(KK),Y(KK),G(I,J))
   H(I,J) = 3.1415926
30 CONTINUE
   ...
   RETURN
   END
```

The EXTINPC subroutine is also called by the INTERPC subroutine to compute the potential and the fluxes at internal points.

The code presented defines some global variables and seven functions. Those variables and functions can be gathered in a *class*, as shown below :

```
class poconbe {
  int N,L;
  double X[101],Y[101],XM[100],YM[100];
  int    KODE[100];
  double FI[100],DFI[100];
  double G[100][100],H[100][100];
  double CX[20],CY[20],POT[20],FLUX1[20],FLUX[20];
  void   Extin(double,...,int);
  void   Locin(double,...,double&);
public:
  poconbe(void);
  void Input(void);
  void GHmat(void);
  void SLNPD(void);
  void Inter(void);
  void Output(void);
}
```

Notice the functions *Extin* and *Locin* declared as *private* members of the *class*. These functions are called only by other member functions and can be *encapsulated*.

The main body of the program would be :

```
void main(void) {
  poconbe object;
  object.Input();
  object.GHmat();
  object.SLNPD();
  object.Inter();
  object.Output();
}
```

The *class constructor* must be defined using the *scope resolution operator (::)* :

```
poconbe::poconbe(void) {
  N = L = 0;
  for (int i=0;i<100;i++)
   { X[i] = Y[i] = FI[i] = 0.0; KODE[i] = 0; }
  for (i=0;i<20;i++) CX[i] = CY[i] = 0.0;
}
```

All other member functions of the *class* can be defined in the same fashion. The simple translation of the code inside of the functions is beyond of the scope of this work.

3.2 POTENTIAL PROBLEMS USING LINEAR ELEMENTS

The second code reviewed solves potential problems using linear elements. The size of some matrices are different from the previous case, the XM and YM arrays are no longer needed and the role of the FI and DFI arrays are changed, as shown :

```
  PROGRAM POLINBE
  DIMENSION X(81),Y(81),FI(80),DFI(160)
  DIMENSION KODE(160),CX(20),CY(20),POT(20),FLUX1(20),FLUX(2)
  COMMON/MATG/ G(80,160)
  COMMON/MATH/ H(80,80)
  COMMON       N,L
C READ DATA
  CALL INPUTPL(CX,CY,X,Y,KODE,DFI)
C COMPUTE THE H AND G MATRICES AND FORM SYSTEM ( A X = F )
  CALL GHMATPL(H,FI,D,160)
C SOLVE SYSTEM OF EQUATIONS
  CALL SLNPD(H,FI,D,80)
C COMPUTE THE POTENTIAL AND FLUXES AT INTERNAL POINTS
  CALL INTERPL(FI,DFI,KODE,CX,CY,X,Y,POT,FLUX1,FLUX2)
C PRINT RESULTS AT BOUNDARY NODES AND INTERNAL POINTS
  CALL OUTPUTPL(X,Y,FI,DFI,CX,CY,POT,FLUX1,FLUX2)
  STOP
  END
```

Again, the program variables and functions can be gathered in a *class*, as shown :

```
class polinbe {
  int N,L;
  double X[81],Y[81];
  int    KODE[160];
  double FI[80],DFI[160];
  double G[80][160],H[80][80];
  double CX[20],CY[20],POT[20],FLUX1[20],FLUX[20];
  void   Extin(double,...,int);
  void   Locin(double,...,double&);
public:
  polinbe(void);
  bool Input(void);
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(void);
}
```

Notice the naming of the functions in the *class* above and in the *class* poconbe, shown in the previous section : related purpose functions have the same name.

3.3 ELASTOSTATICS PROBLEMS USING CONSTANT ELEMENTS

This section reviews a computer code for the solution of two-dimensional isotropic elastostatics problems without body forces. The code has a similar organization as those described previously and the variables and functions used in the program are listed below :

```
      PROGRAM ELCONBE
      DIMENSION X(51),Y(51),XM(50),YM(50),FI(100),DFI(100)
      DIMENSION KODE(100),CX(20),CY(20),SSOL(60),DSOL(40)
      COMMON/MATG/ G(100,100)
      COMMON/MATH/ H(100,100)
      COMMON       N,L,M,GE,XNU,NC(5)
C     READ DATA
      CALL INPUTEC(CX,CY,X,Y,KODE,FI)
C     COMPUTE THE H AND G MATRICES AND FORM SYSTEM (A X = F)
      CALL GHMATEC(X,Y,XM,YM,G,H,FI,DFI,KODE,100)
C     SOLVE SYSTEM OF EQUATIONS
      CALL SLNPD(G,DFI,D,2*N,200)
C     COMPUTE STRESS AND DISPLACEMENTS AT INTERNAL POINTS
      CALL INTEREC(FI,DFI,KODE,CX,CY,X,Y,SSOL,DSOL)
C     PRINT RESULTS AT BOUNDARY NODES AND INTERNAL POINTS
      CALL OUTPUTEC(XM,YM,FI,DFI,CX,CY,SSOL,DSOL)
      STOP
      END
```

As in the previous codes, additional subroutines are called from another part of the program and a new *class* can be declared for the new case :

```
class elconbe {
  int N,L,M;
  double X[51],Y[51], XM[50],YM[50];
  int    KODE[100],NC[5];
  double FI[100],DFI[100];
  double GE,XNU;
  double G[100][100],H[100][100];
  double CX[20],CY[20],DSOL[20],SSOL[20];
  void   Extin(double,...,double&);
  void   Locin(double,...,double&);
  void   Sigma(double,...,double&);
public:
  elconbe(void);
  bool Input(void);
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(void);
}
```

3.4 SOME FIRST PRACTICAL REMARKS

Every code previously presented defines some global variables and functions. In each program, those variables and functions can be gathered in *classes*, specific to each problem but similar in its declarations, as shown below :

```
class boundary {
  ...
  // private variables
  ...
  void Extin(...);
  void Locin(...);
public:
  boundary(void);
  bool Input(void);
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(void);
}
```

It must be pointed out that the elastostatics code declares an additional function, *Sigma*.

The main body of the program is the same in all cases and the only change refers to the type of the object :

```
void main(void) {
  classname object;
  if (object.Input()) return;
  object.GHmat();
  if (object.SLNPD()) return;
  object.Inter();
  object.Output();
}
```

By simply applying the basic concepts of this programming technique, different codes with increasing complexity can be written in the same fashion.

The use of classes allows the programmer to use the same name to declare related functions and to avoid using global variables and its accidental modification or incorrect use.

Besides the safety factor, creating a generic programming "interface" to different applications by itself justifies the use of Object Oriented Programming.

3.5 APPLYING POLYMORPHISM

The concept of *polymorphism* can be applied to improve the first code presented, Potential Problems using Constant Elements. The input function defined in the *class* can be *overloaded* to allow data input from the keyboard or from a text file. The output function can also be *overloaded* to allow result output to the screen or into a text file, as shown below :

```
class poconbe {
  int N,L
  double X[101],Y[101], XM[100],YM[100];
  int    KODE[100];
  double FI[100],DFI[100];
  double G[100][100],H[100][100];
  double CX[20],CY[20],POT[20],FLUX1[20],FLUX[20];
  void   Extin(double,...,int);
  void   Locin(double,...,double&);
public:
  poconbe(void);
  bool Input(void);  // keyboard
  bool Input(char*); // text file
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(void); // screen
  void Output(char*);// text file
}
```

The *overloaded* functions are defined in the same way as any member function :

```
poconbe::Input(void) {
  ...
  // code to read the data from keyboard
  ...
}
poconbe::Input(char *filename) {
  ...
  // code to read data from the text file
  ...
}
poconbe::Output(void) {
  ...
  // code to display results on screen
  ...
}
poconbe::Output(char *filename) {
  ...
  // code to write resuls to the text file
  ...
}
```

The main body of the program developed to use the keyboard and the screen is the same already presented :

```
void main(void) {
  poconbe object;
  if (object.Input()) return;
  object.GHmat();
  if (object.SLNPD()) return;
  object.Inter();
  object.Output();
}
```

The text file version of the same program would be :

```
void main(void) {
  char inputfile[12],outputfile[12];
  cout << "Input file name : ";
  cin >> inputfile;
  cout << "Output file name : ";
  cin >> outputfile;
  poconbe object;
  if (object.Input(inputfile)) return;
  object.GHmat();
  if (object.SLNPD()) return;
  object.Inter();
  object.Output(outputfile);
}
```

The compiler selects the correct function depending on the type of data by which it is called.

3.6 APPLYING INHERITANCE

The concept of *inheritance* can be applied here to create a new *class* from an already defined one, declaring only what is specific to the new *class*.

Consider the *class* poconbe already presented in Potential Problems using Constant Elements. The *class* presented previously is applicable to problems with only one surface but by using the *inheritance* mechanism it can be used as a *base class* to create a new one to solve multiboundary problems.

```
class poconbe {
protected:
  int N,L;
  double X[101],Y[101], XM[100],YM[100];
  int    KODE[100];
  double FI[100],DFI[100];
  double G[100][100],H[100][100];
  double CX[20],CY[20],POT[20],FLUX1[20],FLUX[20];
  void Extin(double,...,int);
  void Locin(double,...,double&);
public:
  poconbe(void);
  bool Input(void);
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(void);
}
```

Notice the change of declaration of variables from *private* to *protected*. This change allows its access by *derived classes* while they remain inaccessible to the rest of the program.

3.7 MULTIBOUNDARY POTENTIAL PROBLEMS USING CONSTANT ELEMENTS

The next code reviewed solves multiboundary potential problems and is based on the constant element code. The program defines an additional variable to hold the number of different surfaces and an array to store the last node of each different surface. To take into account the different surfaces, three functions are declared :

INPUMPC	: same input required by inputpc plus the surface data
GHMAMPC	: expanded from ghmatpc to differentiate the points on each of the surfaces
INTEMPC	: varies from a few statements from interpc to compute the different surfaces

All other variables and functions are the same as declared in Potential Problems using Constant Elements.

This new *class* is *derived* from the *class* poconbe, having in its declaration only what differs one *class* from the other, as follows :

```
class pomcobe : public poconbe {
  int M;
  int NC[5];
public:
  pomconbe(void);
  bool Input(void);
  void GHmat(void);
  void Inter(void);
}
```

The *constructor* of the *class* initializes only its own input variables :

```
pomcobe::pomcobe(void) {
  M = 0;
  for (int i=0;i<5;i++) NC[i] = 0;
}
```

The main body of the program is similar to the previous codes :

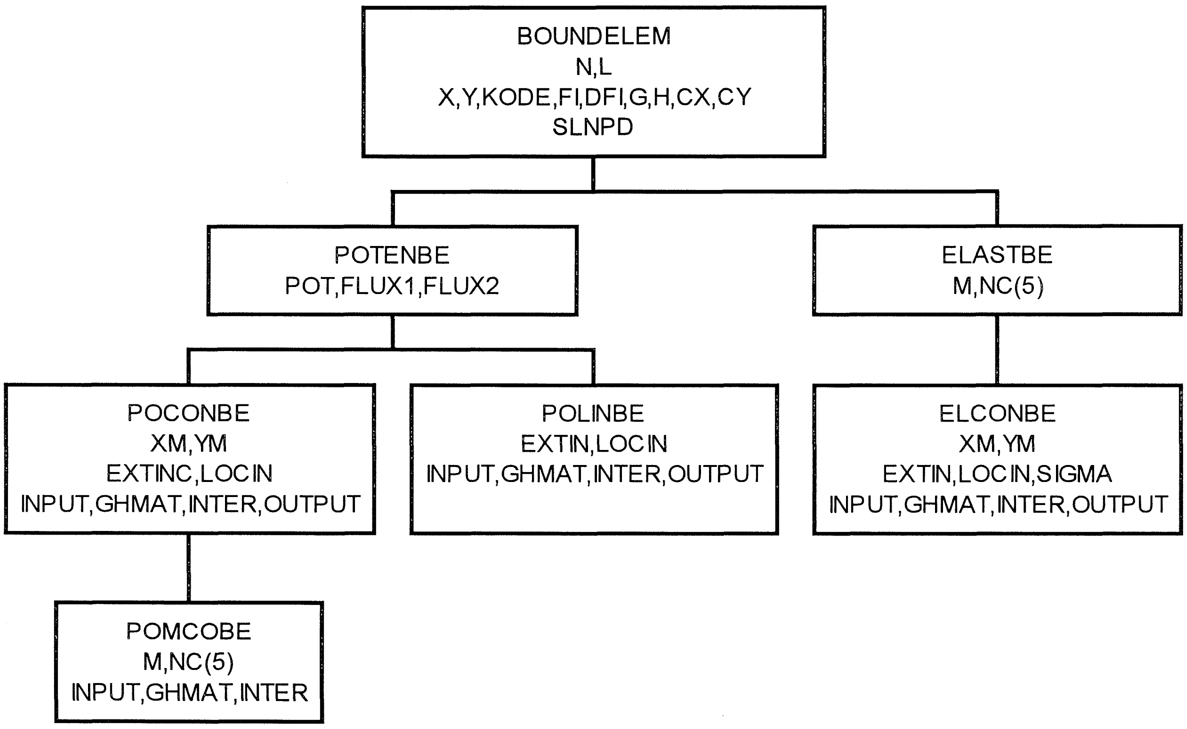
```
void main(void) {
  pomcobe object;
  if (object.Input()) return;
  object.GHmat();
  if (object.SLNPD()) return;
  object.Inter();
  object.Output();
}
```

When this code is executed, the compiler selects the functions defined in the new *class*. Only when they are not redefined in the *derived class*, the functions in the *base class* are executed.

3.8 BOUNDARY ELEMENTS GENERIC CLASSES

The knowledge of function *overloading* and *inheritance* allows a revision of the *classes* previously presented, as follows :

FIG.1 BOUNDARY ELEMENTS GENERIC CLASSES



3.8.1 THE BASE CLASS

The *class* boundelem is a general *base class* to all applications already shown and holds all common variables used in those programs. This *class* also declares a *constructor*, a *destructor* and a function to solve the system of equations, common to all *derived classes*.

```
class boundelem {
protected:
  int N, L;
  double *X,*Y;
  int    *KODE;
  double *FI,*DFI;
  double *G,*H;
  double CX[20],CY[20];
public:
  boundelem(void);
  ~boundelem(void);
  bool SLNPD(void);
}
```

The *constructor* and *destructor* functions of the *class* would be :

```
boundelem::boundelem(void) {
  N = L = 0;
  X = Y = FI = DFI = G = H = NULL;
  KODE = NULL;
  for (int i=0;i<20;i++)
    CX[i] = CY[i] = 0.0;
}
boundelem::~boundelem(void) {
  if (X) delete []X;
  if (Y) delete []Y;
  if (KODE) delete []KODE;
  if (FI) delete []FI;
  if (DFI) delete []DFI;
  if (G) delete []G;
  if (H) delete []H;
  X = Y = FI = DFI =G = H = NULL;
  KODE = NULL;
}
```

The *constructor* of the *class* initializes the variables and pointers to matrices while the *destructor* is used to free the memory allocated to those matrices.

3.8.2 FIRST DERIVATION

The *class* boundelem can be used to *derive* two new general *base classes*, each one with specific elements to solve potential and elastostatics problems :

```
class potenbe : public boundelem {
protected :
  double POT[20],FLUX1[20],FLUX2[20];
public :
  potenbe(void);
}
```

```
class elastbe : public boundelem {
protected:
  int M;
  int NC[5];
  double DSOL[20], SSOL[20];
public :
  elastbe(void);
}
```

The *classes* potenbe and elastbe are *derived* from the *class* boundelem and *inherit* all variables as well as the function already declared in the *base class*. Each new *class* also declares some common variables and its own *constructors* :

```
potenbe::potenbe(void) {
  for (int i=0;i<20;i++)
  POT[i] = FLUX1[i] = FLUX2[i] = 0.0;
}
elastbe::elastbe(void) {
  M = 0;
  for (int i=0;i<5;i++) NC = 0;
  for (i=0;i<20;i++)
    DSOL[i] = SSOL[i] = 0.0;
}
```

There is no need for a *destructor* function once the matrices in those classes are not dynamic allocated.

3.8.3 SECOND DERIVATION

The boundary problems can be solved using different types of elements. The *classes* potenbe and elastbe can be used to *derive* specific *classes* to deal with each one of those cases.

Potential classes :

In those *classes* are declared all the functions and variables specific to solve potential problems using constant and linear elements.

```
class poconbe : public potenbe {
protected :
  double *XM,*YM;
  void Extin(...);
  void Locin(...);
public:
  poconbe(void);
  ~poconbe(void);
  bool Input(void);
  void GHmat(void);
  void Inter(void);
  void Output(void);
}
```

```
class polinbe : public potenbe {
  void Extin(...);
  void Locin(...);
public:
  bool Input(void);
  void GHmat(void);
  void Inter(void);
  void Output(void);
}
```

The two *classes* above are *derived* from the *class* potenbe and *inherit* the variables declared in the *base class* potenbe and its *parent class* boundelem. Only the *class* poconbe has *constructor* and *destructor* functions :

```
poconbe::poconbe(void) {
  XM = YM = NULL;
}
poconbe::~poconbe(void) {
  if (XM) delete []XM; XM = NULL;
  if (YM) delete []YM; YM = NULL;
}
```

Elastostatics class :

In this *class* are declared all the functions and variables specific to solve elastostatic problems using constant elements.

```
class elconbe : public elastbe {
  double *XM,*YM;
  void Extin(...);
  void Locin(...);
  void Sigma(...);
public:
  elconbe(void);
  ~elconbe(void);
  bool Input(void);
  void GHmat(void);
  void Inter(void);
  void Output(void);
}
```

The *class* elconbe is *derived* from the *class* elastbe. It declares an extra *private* function, *Sigma*, additional variables and has *constructor* and *destructor* functions, shown ahead :

```
elconbe::elconbe(void) {
  XM = YM = NULL;
}
elconbe::~elconbe(void) {
  if (XM) delete []XM; XM = NULL;
  if (YM) delete []YM; YM = NULL;
}
```

3.8.4 THIRD DERIVATION

Successive *derivations* can be applied in order to create *classes* with specific behavior. The *class* poconbe has all the implementation necessary to solve Potential Problems using Constant Elements but despite the fact of being a complete *class* from which an *object* can be created, it can be used to *derive* a new class that implements the extra code required to solve multiboundary problems :

```
class pomcobe : public poconbe {
  int M;
  int NC[5];
public:
  pomcobe(void);
  ~pomcobe(void);
  bool Input(void);
  void GHmat(void);
  void Inter(void);
}
```

This *class* declares only the variables not present in its *parent classes* and the functions specific to multiboundary problems. When not *redefined* in the *derived class* pomcobe, the functions declared in the *base class* poconbe will be executed.

The *constructor* and *destructor* functions of the *class* would be :

```
pomcobe::pomcobe(void) {
  M = 0;
  for (int i=0;i<5;i++) NC[i] = 0;
}
pomcobe::~pomcobe(void) {
  if (NC) delete[]NC; NC = NULL;
}
```

4. INTERNET ESSENTIALS

This chapter provides an overview of the *client/server* communication model used in Internet applications and a basic understanding of the networking *protocols*. It also presents the *Common Gateway Interface (CGI)*[6] and the *Java*[15] programming language, which are used in the development of *server* and *client* Internet applications. The last section introduces two programming options to the development of *client* applications specific to the Windows environment.

4.1 THE INTERNET AND THE CLIENT/SERVER MODEL

The Internet is the platform on which the first widely distributed computing applications were developed and deployed. The communication models used in *client/server* computing model of today, where applications processing is distributed over two or more computers, were first developed to accommodate the Internet[16].

The *client/server* model divides application programs into two parts : the *client* program by which the user interacts and the *server* program that provides some kind of services to the *clients*. In the simplest case, these two programs can run on the same computer. More commonly, a local area network (LAN) or the Internet connects the *client* and *server* computers and the application can run regardless the physical location of the hardware.

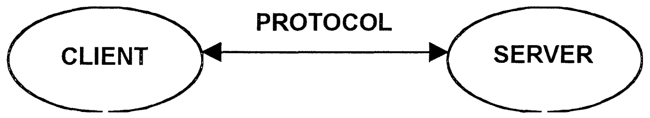
This communication model is quite elegant in its simplicity. First, the *client* connects to the *server*. Next, the *client* and the *server* hold a conversation in which the *client* sends a request to the *server* and the *server* answers with the appropriate reply to the request. This conversation continues for as long as the *client* has requests to make. After the *client* and *server* finish their conversation, the *client* disconnects from the *server* and the session ends.

Most of time, *server* programs run continuously and are typically started by the operating system at boot time and continue to provide services while the machine is running. *Client* programs are launched at the request of the user.

4.2 INTERNET PROTOCOLS

Client/server applications are based on common specifications agreed in advance, known as *protocols*[9]. Computers communicate through streams of data and *protocols* define the format of those streams of data.

FIG.2 THE CLIENT/SERVER MODEL



Protocols are necessary to ensure that all applications will know how to communicate with each other. If the *client* adheres to a given *protocol* while sending the data, any *server* that understands the *protocol* is able to receive and interpret the stream of data. When every application adheres strictly to defined *protocols*, computer can communicate easily, regardless of location, operational system or even hardware.

The open design of the *protocol* specifications gives Internet applications independence from the method used to carry data and thus enables a wide variety of *clients* to access a standard *server*. When all programs use a common specification they can interoperate, although the programs are from different software developers. *Client* programs using different computers and operating systems can also connect to the same *server* if both the *client* and *server* programs use the same *protocol*.

The Internet, like any network, is based on *protocol* layers. Each layer gathers related networking functions and performs specific tasks and services. Each layer is responsible only for a limited set of functions and passes on any other required functions to other layers. As data passes down the *protocol* stack, each layer adds control information - a header - to ensure proper delivery.

FIG.3 THE INTERNET PROTOCOLS LAYERING

Application layer	FTP	HTTP	DNS
Transport layer	TCP		UDP
Internet layer	IP		

4.2.1 THE TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL (TCP/IP)

Almost any data communications network transmits small groups of data, usually referred to as *packets*. A *packet* is simply a specific number of bytes – from 40 to 3200, usually under 1500 bytes - that are grouped together and sent at one time.

The *Internet Protocol*[16] is the foundation of the Internet and provides the most basic level of service, routing a *packet* across the network. *Packet* delivery is *IP*'s one and only job. The *protocol* simply throws *packets* in the network, trying its best to transmit data through the Internet. The *Internet Protocol* provides *best effort* delivery service but does not guarantee that the *packet reaches* its destinations. For instance, *IP* may lose the packet somewhere in the network. This event is usually caused by some sort of network fault. *IP* may also deliver the *packet* corrupted. Although *IP* never intentionally corrupts data en route, it does not check to ensure that the data is not corrupted. If corruption does occur, *IP* will not notice it and will deliver the data anyway. Given a set of *packets*, *IP* does not guarantee that they will arrive in the same order that they were sent. *IP* may choose to send different messages on different paths to the destination, sometimes for the purpose of load balancing and sometimes to avoid a network fault. Because the separate paths may have different delays associated with them, the messages can arrive out of order at the final destination. In all of these cases, *IP* does not do anything to inform either the source or the destination of the *packet* that anything has gone wrong.

One of the key parameters given to *IP* along with a *packet* of data is the destination address for the *packet*. Every device connected to the Internet has a unique 32-bit identifier, its *IP address*. Before a *packet* is sent, *IP* prepends it with a header that contains both the source and destination *addresses* for the *packet*.

In summary, *IP* is an unreliable *protocol* and has no error detection or recovery. That does not mean that one cannot rely on *IP* to deliver data accurately but simply that the *protocol* does not check whether the data was correctly received. When reliable delivery is required other *layers* of the *TCP/IP protocol* provide it.

While *IP* carries a *packet* across the Internet, the *Transmission Control Protocol (TCP)* makes sure that the data inside the *packet* is delivered to its destination safely.

The *TCP protocol* is responsible for error-free data transfer and provides error detection and recovery. While *IP* does not detect errors in the *packet*, *TCP* has a checksum to detect whether a *packet* has been garbled in transmission. If an error occurs, *TCP* provides reliability by resending data until it receives a positive acknowledgment from the remote system. Once *IP* does not guarantee that *packets* will arrive in the same order that they were sent, *TCP* uses a sequencing mechanism to ensure that it delivers data in the proper order. *TCP* automatically reassembles the *packets* into the correct order on the receiving end. If a *packet* fails to arrive because a transmission error, *TCP* resends the *packet* until it has been received.

TCP attaches a header to every *packet*. The information contained in the header is used primarily to detect any transmission error, to reassemble the *packets* on the order end and to deliver data to the correct application.

After routing data through the Internet and delivering the data to the proper destination, the transport *protocol* must pass the data to the correct application process. These application processes, also known as *network services*, are tracked with *port* numbers – 16-bit values that uniquely identify each service. The source *port* number identifies the process that sent the data and the destination *port* number identifies the process that should receive the data. Without *TCP ports* two Internet-connected computers could only have a single conversation at a time. Well-known *ports* provide the consistency needed for a *client* program to access the appropriate program on a *server*. Because the sender and receiver agree in advance which services are offered on which *ports*, the well-known *ports* facilitate the process. With the unique combination of an *IP* address and a *TCP port* number, called a *socket*, a *client* application can establish communications with the exact *server* process.

The Internet protocol suite actually includes other protocols, such as the *User Datagram Protocol (UDP)* and the *Internet Control Message Protocol (ICMP)*. The *User Datagram Protocol* is also an unreliable protocol. *UDP* is used to transmit small amounts of data, to transmit a query and to expect a response within a fixed time period and when a certain amount of *packet* corruption or loss can be accepted. The *Domain Name Server (DNS)* is built on top of *UDP* or *TCP*. The *Internet Control Message Protocol* relies on *IP* to provide diagnostic and control services. The *ping* utility relies on this protocol to check if a host is operational.

4.2.2 THE FILE TRANSFER PROTOCOL (FTP)

The *File Transfer Protocol (FTP)* was one of the first Internet services. *FTP* allows users to transfer files between computers. Part of the purpose for creating the *FTP protocol* was to allow the interchange of files from one system to another while shielding the user from any differences in the underlying file structure of the two (or more) system involved and providing platform independence.[2]

The basic *FTP* communication model uses two communication connections between the *client* and *server* system. The first communication connection passes command and replies between the *client* and *server*. This command connection is open throughout the *FTP* session and passes messages both ways. The second communication connection transfers files and other data from one system to the other. It is open only when transferring data from one computer to the other and passes data in one direction only.

The first connection, the command connection, is initiated by the *FTP client*. The *client* connects to the *server* on *TCP port* 21, gives the *server* a login and a password and then proceeds with the *TCP* session. Most of the simple communications between the *client* and the *server* take place on this first socket connection. If the *client* issues a simple command that requires a single-line response for the *server*, this command connection transmits the reply. When the *client* issues a command that requires more than a single, simple response, such as a directory listing or request to send or receive a file, the second communication connection is used. To establish this second connection, the *server* creates, by default, a socket on *TCP port* 20 and connects to a second socket on the *client* using the same address and port as the first socket on the *client*. The *client*, however, can specify a different address and port to be used for the data transfer.

When transferring data between systems, one can use four data types. Of these, only two are commonly used today, although the other two should still be supported. The ASCII (American Standard Code for Information Interchange) is the default for all *FTP* sessions. It is intended for use when transferring text files. The Image data type is the one most commonly used to transfer binary files such as images and application files. All *FTP* implementations should support this data type as well as the ASCII data type.

4.2.3 THE HYPERTEXT TRANSFER PROTOCOL (HTTP)

The *Hypertext Transfer Protocol (HTTP)* is the *protocol* that supports the most powerful service on the Internet, the *World Wide Web*. *HTTP* is based on the *TCP/IP protocol*. This means that *HTTP* uses *TCP/IP* to transmit data over the Internet. *HTTP* is a higher level *protocol* that defines how the data transmitted by *TCP/IP* should be formatted.

4.2.3.1 THE HYPERTEXT MARKUP LANGUAGE (HTML)

The *HTTP protocol* acts as a generic transport to carry many types of information from the *server* to the *client*. The most common type of data carried across *HTTP* is the *Hypertext Markup Language (HTML.)*[14]. HTTP is a description language on which the www documents are written. In addition to including basic text-formatting directives, HTML has also other directives that provide capabilities such as hypertext links and image loading. Although *HTTP* is used mostly to transmit *HTML* files over the Internet, it can be used to transmit several types of data.

The following example shows the basic structure of an HTML file :

```
<HTML>
  <HEAD>
    <TITLE>
      Document title
    </TITLE>
  </HEAD>
  <BODY>
    <H1>Document heading</H1>
    <P>This is the <B>first</B> paragraph</P>
    <P>This is <I>another</I> paragraph</P>
    <P><A HREF="http://www.ufpr.br">UFPR</A></P>
    <P><IMG SRC="image.gif"></P>
  </BODY>
</HTML>
```

As this example illustrates, *HTML* relies on *tags*, special commands enclosed in angle brackets, to indicate the context and format of the text of a document. *HTML* offers several tags to format text and to display special characters. The linking to other documents is made by including a *tag*. In order to reference images *HTML* provides a *tag* with several options for aligning or positioning.

4.3 THE COMMON GATEWAY INTERFACE (CGI)

The *COMMON GATEWAY INTERFACE (CGI)*[6] is a standard for interfacing applications with web-*servers*. *CGI* programs are applications placed in a special directory in the *server* and executed by a web-*server*, usually in response to a user request from an Internet browser.[3]

CGI programs can be developed in a variety of languages like C/C++, Delphi, Visual Basic and Perl. *CGI* also can be implemented on many different platforms, including Unix and Windows. This work emphasizes C/C++ because of their execution speed, small executable sizes and availability of compilers.

Consider the simple code :

```
#include <iostream.h>
void main(void) {
  cout << "Content-type: text/html\n\n";
  cout << "<HTML>\n";
  cout << "<BODY>\n";
  cout << "<P>Hello Internet World</P>\n";
  cout << "</BODY>\n";
  cout << "</HTML>\n";
}
```

The code above shows a simple C++ program that displays strings on the screen. Like any other program, it can be started by the user and the result is simply a text output. However, the program can be placed in a special directory on a web-*server* and executed by the client browser. In this case, the output is redirected by the web-*server* to the browser of the client.

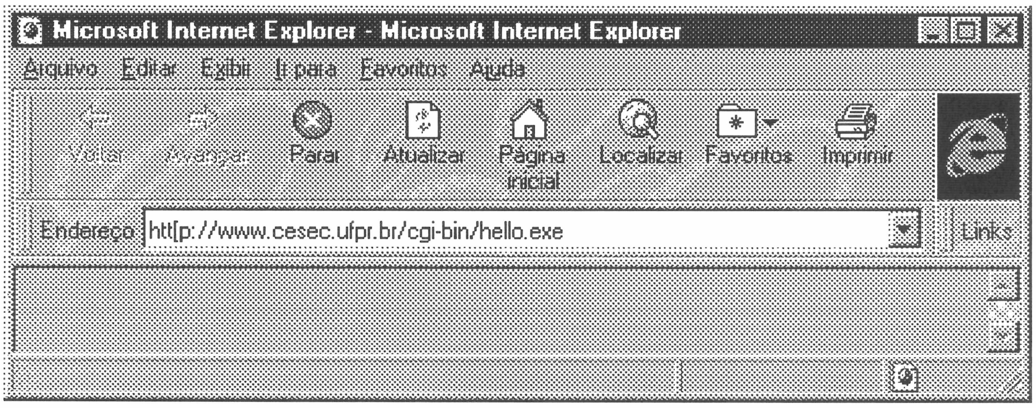
The first line of the code above is a message to the *server*. It says that what follows is HTML text. Since the output of the *CGI* program will pass through the *server* to the *client*, the initial message is required to tell the *server* to forward the message on to the *client*. This line will not be sent to the *client* software. Only the lines that follow will go to the *client*.

When this program is executed by a web-server, the *HTML tags* embedded are interpreted, formatted and then the string *Hello Internet World* is displayed on the browser as a standard *HTML* page.

4.3.1. EXECUTING A CGI PROGRAM FROM A WEB PAGE

CGI programs must reside in the *cgi-bin* directory on Unix systems or in the *scripts* directory on Windows systems. To execute a program in a CGI directory the user must type to the browser the address of the *server* name, the CGI directory and the name of the program to be started :

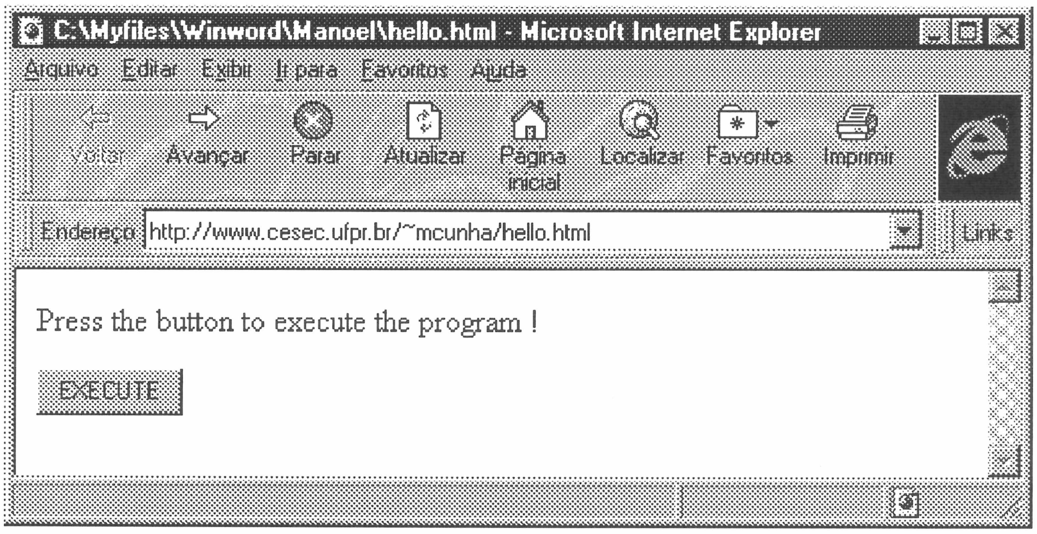
FIG.4 EXECUTING A CGI PROGRAM FROM A BROWSER


Microsoft Internet Explorer - Microsoft Internet Explorer
Arquivo Editar Exibir Ir para Favoritos Ajuda
Voltar Avançar Parar Atualizar Página inicial Localizar Favoritos Imprimir
Endereço
Links

htt[p://www.cesec.ufpr.br/cgi-bin/hello.exe

Although it is possible for starting a *CGI* application directly, the most common way to start a program in a server is from an HTML page. The standard form to start an application is asking the user to press a button on a Web page.

FIG.5 EXECUTING A CGI PROGRAM FROM A WEB PAGE


C:\Myfiles\Winword\Manoel\hello.html - Microsoft Internet Explorer
Arquivo Editar Exibir Ir para Favoritos Ajuda
Voltar Avançar Parar Atualizar Página inicial Localizar Favoritos Imprimir
Endereço
Links

http://www.cesec.ufpr.br/~mcunha/hello.html

Press the button to execute the program !

EXECUTE

The *HTML* page just shown can be coded as follows :

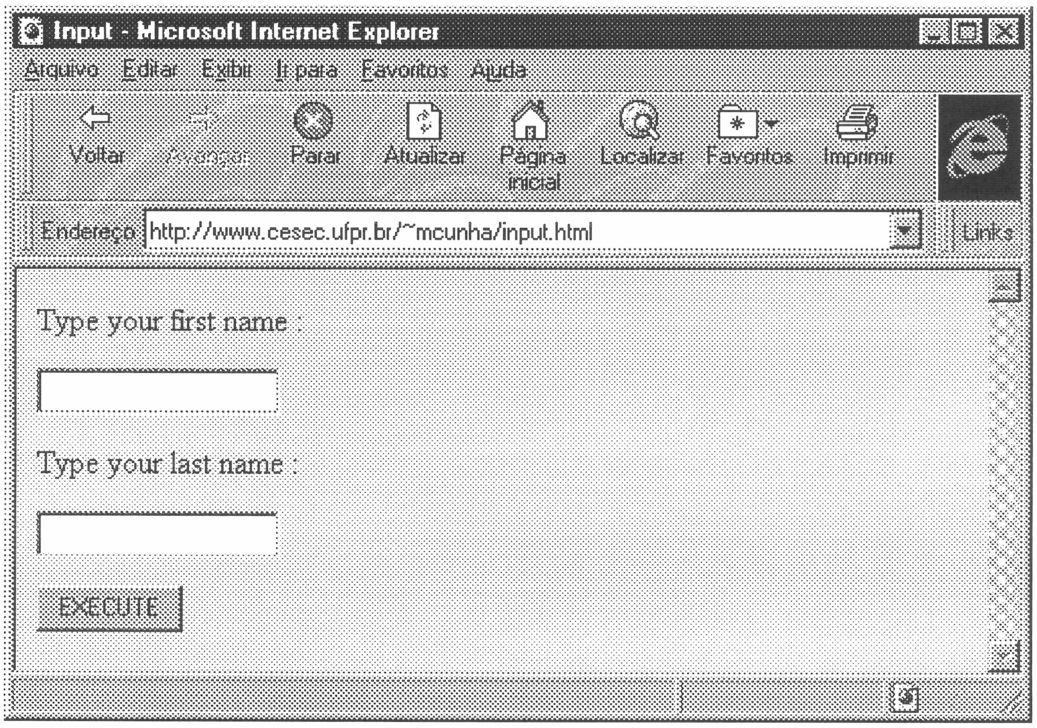
```
<HTML>
  <BODY>
    <FORM ACTION="www.cesec.ufpr.br/cgi-bin/hello.exe">
      <P>Press the button to execute the program !</P>
      <P><INPUT TYPE="submit" VALUE="Execute"></P>
    </FORM>
  </BODY>
</HTML>
```

In the above code, the *<FORM ACTION="address"> tag* informs the location of the *CGI* program to be executed. A button that starts the process is created with the <INPUT TYPE="submit"> *tag*. The text of the button is defined in the *VALUE* parameter.

4.3.2 INPUT FROM A WEB BROWSER

The next step to give a *CGI* application full capabilities to act as a standard application is to provide data input. Using the input types defined by the HTML specification, the *client* browser can start a program placed in the *server* and send to it the data entered by the user in the *HTML* form.

FIG.6 INPUT FROM A WEB PAGE



Consider the following code :

```
<HTML>
  <BODY>
    <FORM ACTION="wwww.cesec.ufpr.br/cgi-bin/input.exe">
      <P>Type your first name :</P>
      <P><INPUT TYPE="text" NAME="firstname"></P>
      <P>Type your last name :</P>
      <P><INPUT TYPE="text" NAME="lastname"></P>
      <P><INPUT TYPE="submit" VALUE="Execute"></P>
    </FORM>
  </BODY>
</HTML>
```

The sample above is similar to the input forms found in many commercial sites like Amazon or search engines like Altavista. At these sites, the user inputs data in an HTML form and presses a button to know more about a book or searching for a subject.

When the user pushes the button the application defined in the *action* line is started and the browser passes the data to it using the following format :

```
firstname=manoel&lastname=cunha
```

All variables are sent as a single and continuous string. Each variable includes a field name and the string value of the variable. There must be an *equal sign (=)* between the field name and the value entered by the user. The variables are separated with *ampersands (&)*. The final variable ends with a *carriage return* character. Spaces can appear as a *plus sign (+)*.

4.3.3 OUTPUT TO A WEB BROWSER

The output of a *CGI* program is very similar to the output to the screen. In fact, the web-*server* redirects the output from the screen to the *client* browser.

Once the web browsers use the *HTML* specification to format data being displayed, the output of a *CGI* application must be appropriately modified. The programmer must include among the output commands, a set of standard *HTML tags* to allow the proper formatting of the data being sent to the *client* browser.

When the *CGI* program is run, environmental variables known to the *server* are provided. Certain environmental variables used by the server can provide valuable information to the *CGI* program. One of these environmental variables can be used by the CGI *program* to receive the string sent by the client browser through the server, as shown below :

```
#include <iostream.h>
#include <stdlib.h>
void main(void) {
  cout << ″Content-type: text/html\n\n″;
  cout << ″<HTML>\n″;
  cout << ″<BODY>\n″;
  cout << ″<P>″ << getenv(″QUERY_STRING″) << ′\n′;
  cout << ″</BODY>\n″;
  cout << ″</HTML>\n″;
}
```

In the above code, the command *getenv* is used with the parameter *"QUERY_STRING"* to retrieve the data string sent by the *client*. The data string is then simply returned to the browser that started the process. To use the information passed to it, the *CGI* program needs only a function to parse each variable of the string.

4.3.4 THE HTTP GET AND POST METHODS :

When the form is sent to the *CGI* program, it contains the user entered information.

Several methods are supported for sending data. The two primary methods are the *Get* and the *Post* ones. The *Get* option is the default if none is stated in the HTML document. The *Get* data arrives as a command line argument or as an environmental variable, as seen before. In the Unix system the *Post* method, data are sent as *standard input (stdin)* while in the Windows system, the user data are passed to a file. This file has a similar format to the initialization files (.ini) used by Windows 3.1x applications.

For portability reasons, all applications presented in this work have been developed using the *Get* method.

4.4 THE JAVA PROGRAMMING LANGUAGE :

While the *CGI* programs provide server-side capabilities, there are many situations where *client*-side capabilities are needed. With the increasing power of the computers used as *client* end, application processing can be distributed between *client* and *server* to take advantage of the cumulative power. In a *Graphic User Interface (GUI)* world, *client*-processing capabilities are needed to give to the user some interactive features not provided by the standard *HTML* specifications.

Java is an Object Oriented Programming language developed by Sun Microsystems in 1991[15]. The first motivation for the development of *Java* was the necessity of a platform-independent language that could be used to create software for consumer electronics. However, with emergence of the Word Wide Web, *Java* was propelled to the forefront of computer languages because the Web, too, demanded portable programs.[25]

Java can be used to create two types of programs : applications and *applets*. An application is a program that runs on a computer, under the operating system of that computer. An application created by *Java* is much as if one created using C or C++. When used to create applications, *Java* is not much different from any other computer language. Rather, the ability of *Java* to create *applets* is what makes it important. An *applet* is an application designed to be transmitted over the Internet and executed by a *Java*-compatible browser. An *applet* is actually a tiny *Java* program, dynamically downloaded across the network, just like an image, sound file or video clip.

4.4.1 THE JAVA APPLETS :

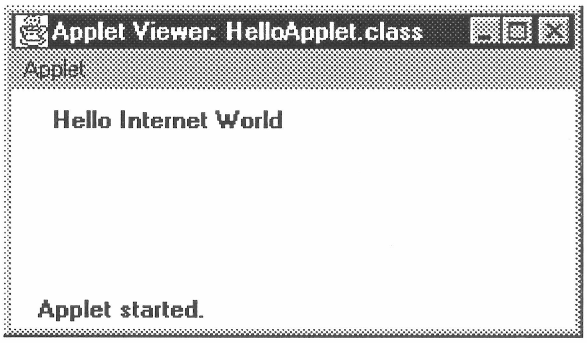
Applets are small applications that are accessed on an Internet *server*, transported over the network, automatically installed and run as part of a Web document.

Consider the simple code :

```
import java.awt.*;
import java.applet.*;
public class HelloApplet extends Applet {
  public void paint(Graphics g) {
    g.drawstring("Hello Internet World",20,20);
  }
}
```

The code above simply displays a message in the *applet* window, as shown :

FIG.7 THE HELLO APPLET



Hello Internet World

Applet started.

Java applets can be executed by a special program called viewer, with debug purpose.

To execute an *applet* in a Web browser, one needs to write a short *HTML* text file that contains the appropriate *applet tag*, as show below :

```
<HTML>
<BODY>
<APPLET CODE="HelloApplet.class" WIDTH=200 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

All *applets* are *derived* from the *Applet class*, which provides the necessary support and gives control over the execution of an *applet*. The *Abstract Window Toolkit (AWT) class* contains support for a window-based, graphical interface.

Applets are event driven, what means that an *applet* waits until an event occurs. The *AWT* notifies the *applet* about an event by calling an event handler. Once this happen, the *applet* must take appropriate action and return control to *AWT*. The most commonly handled events are those generated by the mouse, the keyboard and controls such as a push button.

The AWT supports a rich assortment of graphics functions. All graphics are drawn relative to a window. This can be the main window or a child window of an *applet*. The *Graphics class* defines a number of drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn in the currently selected graphics color. When a graphics object that exceeds the dimensions of the window is drawn, output is automatically clipped. Java supports color in a portable, device-independent fashion. The *AWT* also supports multiple type fonts and the following types of control : labels, push buttons, check boxes, menus, lists, scroll bars and text editing. The *Image class* and the java.awt.image package together provide support for display and manipulation of graphical images. [6]

Parameters can be passed to a Java *applet* with the use of HTML tags. The *getParameter class* function is used to retrieve a parameter in an *applet*, as shown below :

```
import java.awt.*;
import java.applet.*;
/*
<applet code="InputApplet" width=200 height=100>
<param name=firstname value=manoel>
<param name=lastname value=cunha>
</applet>
*/
public class InputApplet extends Applet {
  String firstname;
  String lastname;
  public void start() {
    firstname = getParameter("firstname");
    lastname  = getParameter("lastname");
  }
  public void paint(Graphics g) {
    g.drawString(firstname+" "+lastname,10,10);
  }
}
```

Values returned from *getParameter* must be tested. If a parameter is not valid, *getParameter* will return *null*. In addition, conversions to numeric types must be attempted in a *try* statement, which catches *NumberFormatException*. Uncaught *exceptions* should never occur within an *applet*.

4.4.2 THE JAVASCRIPT

While *Java* is a programming language similar to C and C++, *Javascript*[16] is a script language similar to Unix *shell* or *Perl*[15]. Javascript, unlike *Java*, cannot be used except with a Web browser. However, unlike *Java*, *Javascript* is not checked and is subject to bad coding and errors, usually detected by a compiler. In addition, *Javascript* has none of the security mechanism found in *Java*. *Javascript* also does not support advanced features like graphics capabilities. Despite these limitations, *Javascript* can provide some *HTML* enhancement capabilities for Web pages developers.

4.4.3 JAVA AND CGI PROGRAMS

One of the most important gains for *CGI* programmers is the ability to preprocess the input of the user on the *client* side before sending it to the *server CGI* program. The use of *client*-side preprocessing can allow such functions as range checking or data validation to take place before the data are sent to the *server CGI* program. Because error messages can be generated on the *client* side, the user is given a quicker response to both a valid and an invalid entry. Corrections take place more quickly if data are entered incorrectly.

Although *CGI* and *Java* programming are distinct, these languages can interface in a several basic ways. *CGI* program can send *Java applets* as a part of the HTML sent from the *server* to the *client*. In this instance, the CGI program can be used to change the parameters of an *applet* or even the *applet* itself. Since *Java* includes the capability of sending out a *Universal Resource Locator (URL)*, it is possible to a *Java applet* to start up a *CGI* program.

The use of *Java applets* in combination with CGI programs may become a very complex project. To take full advantage of their capabilities, a programmer must have a deep knowledge of advanced programming techniques like *sockets*, *multithreading* and *exception handling*. *Java* is a language grounded in the needs of experienced programmers and demands great familiarity with Object Oriented Programming.

4.5 DEVELOPING WINDOWS INTERNET APPLICATIONS :

Microsoft provides two sets of Application Programming Interfaces (API) to assist in Windows application development. The first and most widely used in existing applications is known as the *Winsock API*[16]. Although *Winsock* takes care of most of *TCP/IP* details for the programmer, it is still complex and requires the implementation of all functionality needed by an application. The second and much newer *API* is known as the *Windows Internet API or WinInet*[25]. *WinInet* is a much higher level interface to the *HTTP*, *FTP* and *Gopher protocols*. Developers can use *WinInet* to add powerful communication functionality without having to deal with the complexities of the underlying *protocols*.

While *Winsock* can be used to develop both *client* and *server* applications and supports all Internet *protocols*, WinInet are used to develop only *client* applications and supports only three *protocols*, one of them *out-of-use*.

Winsock and *WinInet* allows the development of *client* applications with full capabilities to download Web pages, execute *server* applications and transfer files between computers. Those capabilities can be combined with all the features provided by the Windows environment, in special the graphical interface, to produce highly efficient distributed applications. Web *servers* can be used to provide network-centered processing capabilities as well as they can be used simply to hold shared information. In both cases, *client* applications can be developed to give the user a graphical interface to a *CGI* program or a database. *Client* applications can also be used to preprocessing input data as well as postprocessing output results, going on-line only when data needs to be transmitted and/or processed by a remote *server*.

4.5.1 THE WINDOWS INTERNET API (WININET)

The basic features of the Windows Internet API can be applied to develop a simple application to download an *HTML* file from a Web *server* or a text file from an FTP *server*. For the sake of simplicity, the sample presented is of console type but is implemented with the same capabilities of a conventional Windows application.

Consider the following code :

```
#include "windows.h"
#include "iostream.h"
#include "wininet.h"

void main(void) {
char url[64]="";
cout << "URL : ";
cin >> url;
if (!*url) return;
HINTERNET hSession =
InternetOpen("App/1.0",INTERNET_OPEN_TYPE_PRECONFIG,NULL,NULL,0);
if (hSession) {
  HINTERNET hFile = InternetOpenUrl(hSession,url,NULL,0,0,0);
  if (hFile) {
    char buffer[1024]="";
    DWORD dwRead;
    while (InternetReadFile(hFile,buffer,1023,&dwRead)) {
      if (dwRead == 0) break;
      buffer[dwRead] = 0;
      cout << buffer;
    }
    cout << '\n';
    InternetCloseHandle(hFile);
  }
  else MessageBox(0,"InternetOpenUrl","Error",MB_OK);
  InternetCloseHandle(hSession);
}
else MessageBox(0,"InternetOpen","Error",MB_OK);
cout << "End of program\n";
}
```

In the above code, the user is asked to input a *Universal Resource Locator (URL)*, used to define the file to be downloaded and displayed on the screen. The Internet *protocol* used to transfer the data is defined by the user within the *URL*. For example, if the user type the address `http://www.cesec.ufpr.br/~mcunha/hello.html`, the program uses the *HTTP protocol* to transfer the *hello.html* file. The user can also type the address of a text file in an FTP *server* and the program will show the content of the chosen file in the screen.

The last code presented can easily be changed to redirect the output from the screen to a disk file or to an edit window. With minimum effort, a programmer can also change the application giving it uploading capabilities. The knowledge needed to accomplish the task of developing an Internet Windows application is only a few WinInet functions. However, the Windows API is very complex and its use is out of the scope of this work.

5. INTERNET APPLICATIONS

In Chapter 3 some computer codes are reviewed and rewritten under the paradigm of *Object Oriented Programming*. At that chapter, the main goal is to present its basic concepts and how computer codes written in a traditional and a very conservative fashion can be easily rewritten using this approach. Chapter 4 introduces the *Common Gateway Interface (CGI)* and the *Hypertext Markup Language (HTML)*. With the knowledge of these two programming techniques, a programmer can produce *server*-side applications and use Web browsers as the *client*-side user interface.

This chapter combines all concepts already presented in an engineering context. The computer codes reviewed in Chapter 3 will now be improved using *Object Oriented Programming* mechanisms by making them *CGI* applications and giving them an *HTML* interface.

Each program defines input and output functions. The input function reads all the input data required by each program. The output function is used to present results to the user. Usually the programmer implements first input and output tasks using files. In this way, the development of the application can be focused to the problem to be solved. After the implementation of the solution, the programmer can develop a more efficient or interactive interface to the user.

To transform those programs in *CGI* applications with *HTML* interface, one needs only to implement specific input and output functions. The input function must read the data string supplied by the Web *server* and must parse the information embedded in the string. The output function must use the *HTML* tags to format the results that will be displayed by the client browser.

As shown before, the input function can be *overloaded* to allow data input from different sources like the keyboard or text file. The output function can also be *overloaded* to allow output results to the screen or text file. Using one of the most important features of *Object Oriented Programming*, the *polymorphism*, the programmer needs only to *overload* both functions to accommodate a new option like a browser input and output. Using *polymorphism*, the programmer only increases the capabilities of a program, without any lost of work and time.

5.1 POTENTIAL PROBLEMS USING CONSTANT ELEMENTS

The first code presented is the computer code for solving Laplace type problems using Constant Boundary Elements. The POCONBE *class* created to the first program declares some *private* variables and *public* functions that manipulates that variables, as shown below :

```
class poconbe {
  int N,L;
  double X[101],Y[101],XM[100],YM[100];
  int    KODE[100];
  double FI[100],DFI[100];
  double G[100][100],H[100][100];
  double CX[20],CY[20],POT[20],FLUX1[20],FLUX[20];
  void   Extin(double,...,int);
  void   Locin(double,...,double&);
public:
  poconbe(void);
  bool Input(ifstream); // text file
  bool Input(char);     // web server
  void GHmat(void);
  bool SLNPD(void);
  void Inter(void);
  void Output(ofstream); // text file
  void Output(void);     // web browser
}
```

The main body of the *CGI* version of the program could be :

```
void main(void) {
  poconbe object;
  object.Input(getenv("QUERY_STRING"));
  object.GHmat();
  object.SLNPD();
  object.Inter();
  object.Output();
}
```

As shown before, when a *CGI* application is started by the Web *server*, the input data is sent as a string. In the above code, that data string is passed as a parameter to the Input function. The parse of the data string inside the function is a simple programming task, out of the scope of this work.

The output to a Web browser is very similar to an output to the screen. The only change is the inclusion of HTML tags to format text. Special care must be taken when declaring functions to the screen and to the Web browser once the screen output is redirected by the Web server.

5.1.1 INPUT DATA FROM A WEB BROWSER

A *CGI* application is a program placed in a special place on the *server*. This program is started by the Web server by request of a *client* user using a browser like Netscape or Internet Explorer. Most of the time, the user starts an application and supplies data to that program using a standard form. Web forms are text files formatted accordingly the *HTML* specification. An *HTML* page can contain well known interface resources, like text boxes, push buttons, check boxes, radio buttons, among others. Images and sounds also can be used to enhance this user interface.

The *Poconbe* application must be compiled and placed in the *cgi-bin* directory on Unix systems or in the *scripts* directory on Windows systems. The next step is to create an *HTML* page to be accessed by the user, using a web browser. It must be noticed that the *HTML* file is not placed in the same directory of the *CGI* program. The exact location of the *HTML* directory also depends of system being used.

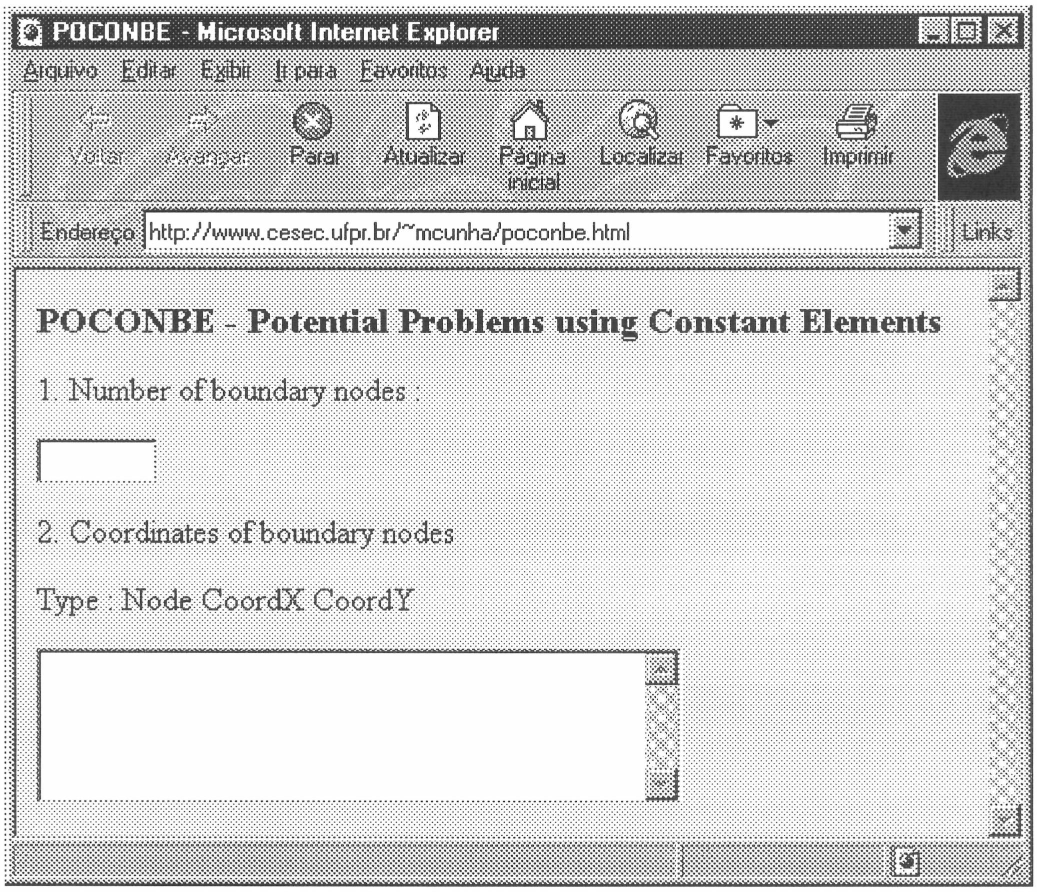
Consider the *HTML* code below :

```
<HTML>
<HEAD><TITLE>POCONBE</TITLE></HEAD>
<BODY>
<H1>POCONBE - Potential Problems using Constant Elements</H1><HR>
<FORM ACTION="http://www.cesec.ufpr.br/cgi-bin/poconbe.exe">
<P>1. Number of boundary nodes :</P>
<INPUT TYPE="text" VALUE="" SIZE=8 NAME="numnodes">
<P>2. Coordinates of boundary nodes</P>
<P>Type : Node CoordX CoordY</P>
<TEXTAREA COLS=64 ROWS=8 VALUE="" NAME="nodecoords"></TEXTAREA>
<P>3. Boundary conditions</P>
<P>Type : Element 0/1 Value</P>
<TEXTAREA COLS=64 ROWS=8 VALUE="" NAME="boundcond"></TEXTAREA>
<P>4. Number of internal points :</P>
<P><INPUT TYPE="text" VALUE="" SIZE=8 NAME="numintpts"></P>
<P>5. Coordinates of internal points</P>
<P>Type : Point CoordX CoordY</P>
<TEXTAREA COLS=64 ROWS=8 VALUE="" NAME="intncoords"></TEXTAREA>
<P><INPUT TYPE="submit" VALUE="EXECUTE"</P>
</FORM>
<BODY>
</HTML>
```

The above code is a *HTML* page that provides an user interface to program *Poconbe*. It can be used to execute the application and pass data to it.

Besides the text used to guide the user in the data input process, this HTML code uses two types of controls to allow that data input.

FIG.8 A WEB INTERFACE TO CGI PROGRAMS


POCONBE - Microsoft Internet Explorer
Arquivo Editar Exibir Ir para Favoritos Ajuda
Voltar Avançar Parar Atualizar Página inicial Localizar Favoritos Imprimir
Endereço http://www.cesec.ufpr.br/~mcunha/poconbe.html
Links

POCONBE - Potential Problems using Constant Elements

1. Number of boundary nodes :

2. Coordinates of boundary nodes

Type : Node CoordX CoordY

The first type of control uses the `<INPUT TYPE="text">` *tag* to display a single-line text box. The second type uses the `<TEXTAREA>` *tag* to a multi-line text box. Both *tags* have parameters to define the size and the name of each control. The name of each control will be passed in the data string as the variable names. The data typed by the user in each control will be sent in the data string after the variable name and separated from it by the equal sign (=). Each variable is separated by an *ampersand (&).*

The *HTML* page has two single lines edit controls, used to input the number of boundary nodes and the number of internal points : the N and L variables of the program. The three multi-line controls are used to input boundary node coordinates, boundary conditions and internal points coordinates : the X, Y, KODE, FI, CX and CY arrays. It is a relative simple task to parse the string and convert it to appropriate variables of the application.

The last control used in the *HTML* page is the *Execute* button, defined with the `<INPUT TYPE="submit">` *tag*. This button must be pressed by the user to execute the *CGI* application. The program to be executed is defined in the *<FORM ACTION="address"> tag*.

5.1.2 OUTPUT RESULTS TO A WEB BROWSER

To output results to a browser is an easy task, very similar to display results to the screen. The only changes are the inclusion of a command line to the *server* and *HTML tags* to format the data being presented on the browser.

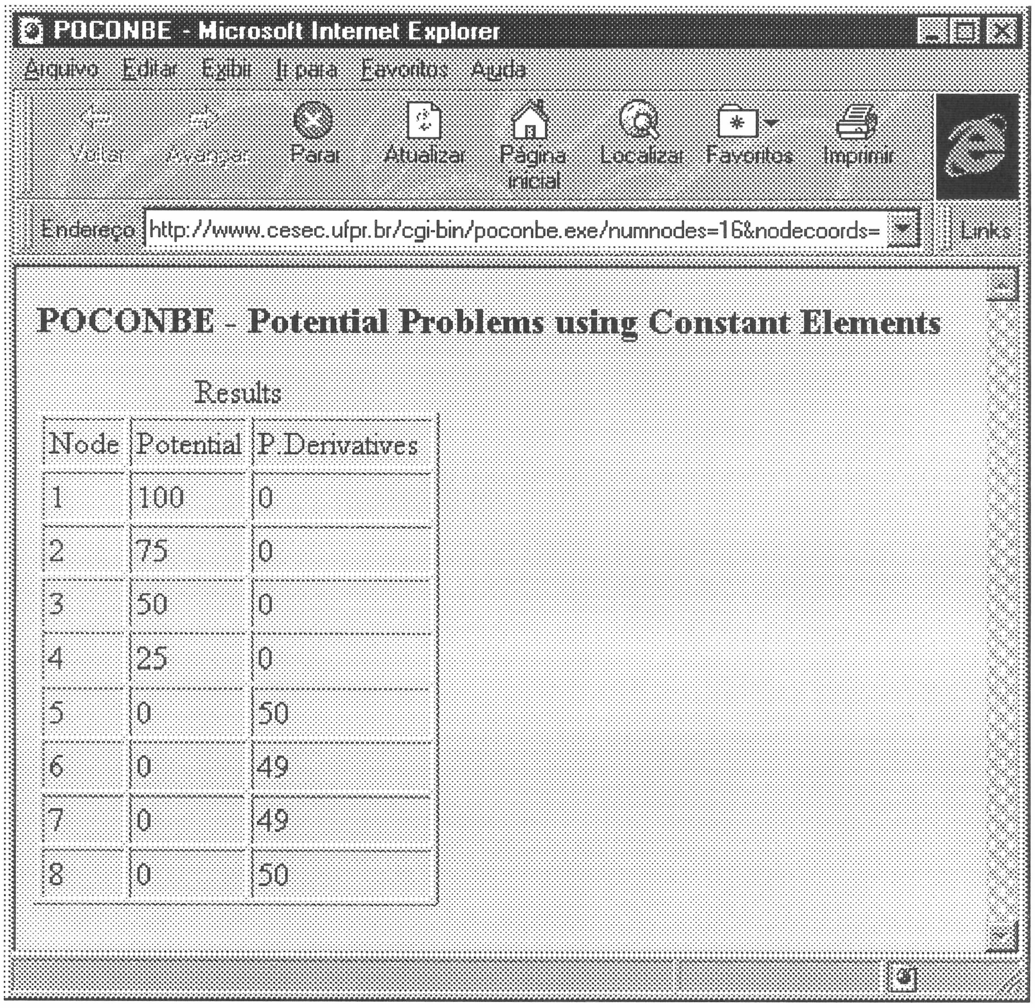
As stated before, the first line of the code must be a message to the *server*. It says that what follows is an *HTML* text. Since the output of the *CGI* program will pass through the *server* to the *client*, the initial message is required to tell the *server* to forward the message on to the *client*. This line will not be sent to the *client* software. Only the lines that follow will go to the *client*.

Consider the following *HTML* code :

```
void Output(void) {
cout << "Content-type:text/html\n\n";
cout << "<HTML>\n";
cout << "<HEAD><TITLE>POCONBE</TITLE></HEAD>\n";
cout << "<BODY>\n";
cout << "<H1>POCONBE - Potential Problems using Constant Elements</H1>\n";
cout << "<TABLE BORDER=1 CELLPADDING=2>\n";
cout << "<CAPTION>Results</CAPTION>\n";
cout << "<TR><TH>Node<TH>Potential<TH>P.Derivatives\n";
for (int i=0;i<N;i++)
  cout << "<TR><TD>" << i+1 << "<TD>" << FI[i] << "<TD>" << DFI[i] << '\n';
cout << "</TABLE>\n";
if (L) {
  cout << "<TABLE BORDER=1 CELLPADDING=2>\n";
  cout << "<CAPTION>Internal Points</CAPTION>\n";
  cout << "<TR><TH>Point<TH>Potential<TH>Flux X<TH>Flux Y\n";
  for (int k=0;k<L;k++) {
    cout << "<TR><TD>" << k+1 << "<TD>" << POT[k];
    cout << "<TD>" << FLUX1[k] << "<TD>" << FLUX2[k] << '\n';
  }
  cout << "</TABLE>\n";
}
cout << "</BODY>";
cout << "</HTML>";
}
```

In the code just presented, *HTML tags* are used to display arrays. The *<TABLE> tag* starts and finishes the output of an array. Each line of the array is defined by the *<TR> tag* and the *<TD> tags* separate each element in the line. One can notice that an *HTML* table can be used to display the elements of more than one array at the same time.

FIG.9 AN OUTPUT FROM A CGI PROGRAM



POCONBE - Microsoft Internet Explorer
Arquivo Editar Exibir Ir para Favoritos Ajuda
Voltar Avançar Parar Atualizar Página inicial Localizar Favoritos Imprimir
Endereço
Links

http://www.cesec.ufpr.br/cgi-bin/poconbe.exe/numnodes=16&nodecoords=

POCONBE - Potential Problems using Constant Elements

Results

Node	Potential	P.Derivatives
1	100	0
2	75	0
3	50	0
4	25	0
5	0	50
6	0	49
7	0	49
8	0	50

A good programming practice is to output the input data back to the user. Another good technique used by experienced programmers is to foresee error occurrence, allowing feedback messages about those errors.

5.2 POTENTIAL PROBLEMS USING LINEAR ELEMENTS

The second code presented solves potential problems using linear elements. The considerations about the declaration of the input and output functions are the same discussed in the previous case. The *HTML* code is almost the same presented before. The only change is the way that the information typed by the user in one of the controls. Linear elements need more information about the element boundary conditions.

The job of parsing the data string and converting it to the program variables still is performed in the input function. The output function only needs to be redefined if the input data must be printed with the results.

5.3 ELASTOSTATICS PROBLEMS USING CONSTANT ELEMENTS

The computer code for the solution of two-dimensional isotropic elastostatics problems without body forces has a similar organization as those described previously. No additional consideration is needed when turning the application presented previously into a CGI application. The creation of an *HTML page* to provide an interface to the user is the same presented before.

5.4 USING IMAGES

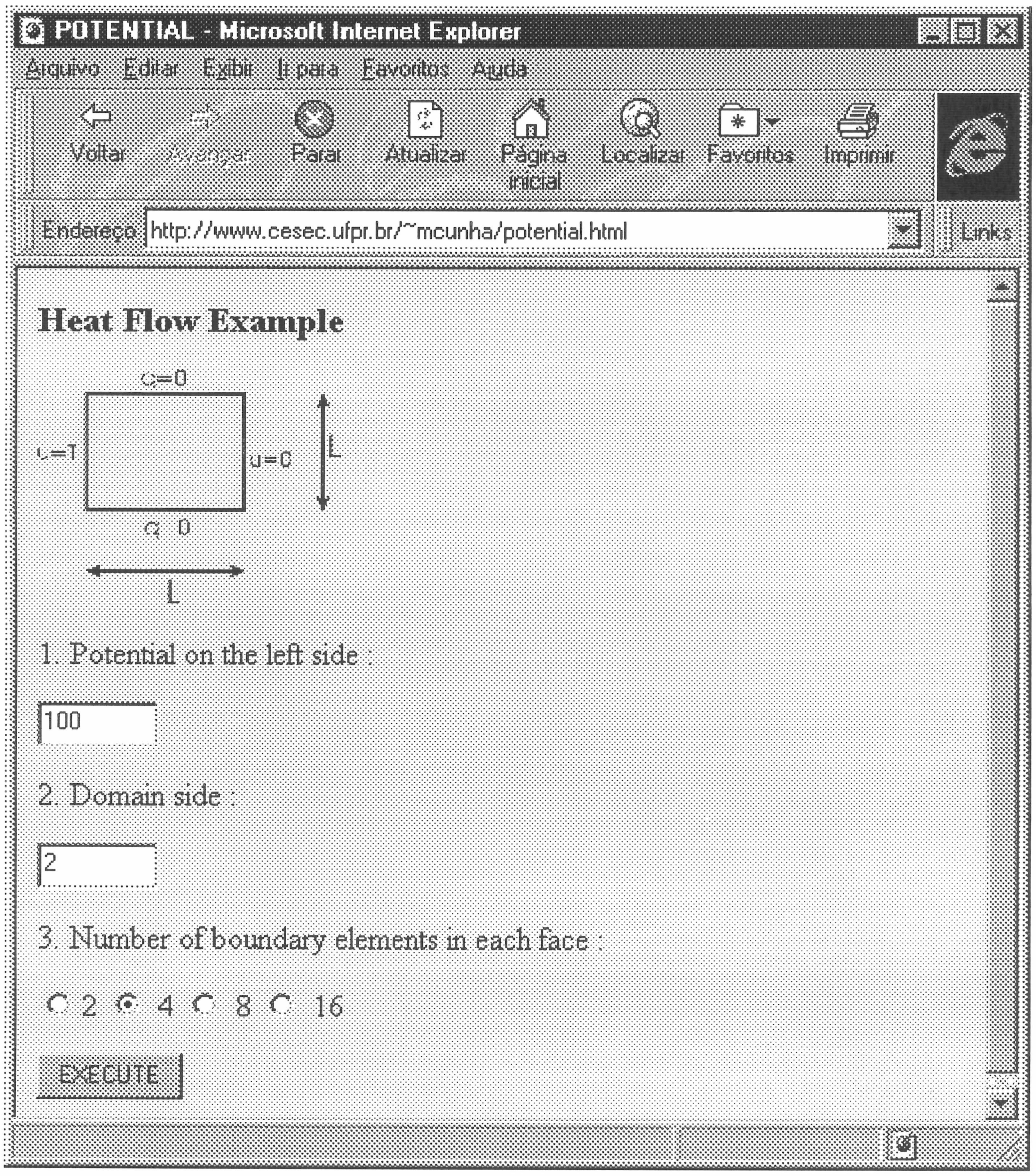
The `<INPUT TYPE="text">` *tag* and the `<TEXTAREA>` *tag* can be used to provide input controls in an *HTML* page. However, the *HTML* capability of displaying images can be used to enhance an *HTML* page, giving to the user a better support in the input process. This feature can also be used to provide a graphic visualization of the results.

To reference images, *HTML* provides the `<IMG SRC="image.gif">` *tag* with several options for aligning and positioning. The *HTML* specifications define that the image files referenced by the *tag* must be of the *GIF* or *JPG* type.

5.4.1 THE POTENTIAL APPLICATION

This application was developed for educational purposes to reproduce one of the examples presented by Brebbia and Dominguez[1]. It illustrates how the first code can be used to analyze a simple potential problem. In this application, the user must define the size of a square domain, the number of elements of each side as well as the temperature at the left side of the domain.

FIG.10 THE POTENTIAL APPLICATION


POTENTIAL - Microsoft Internet Explorer
Arquivo Editar Exibir Ir para Favoritos Ajuda
Voltar Avançar Parar Atualizar Página inicial Localizar Favoritos Imprimir
Endereço
Links
q=0
u=T
u=0
q=0
L
L
EXECUTE

http://www.cesec.ufpr.br/~mcunha/potential.html

Heat Flow Example

1. Potential on the left side :

100

2. Domain side :

2

3. Number of boundary elements in each face :

2 [✓ 4] 8 16

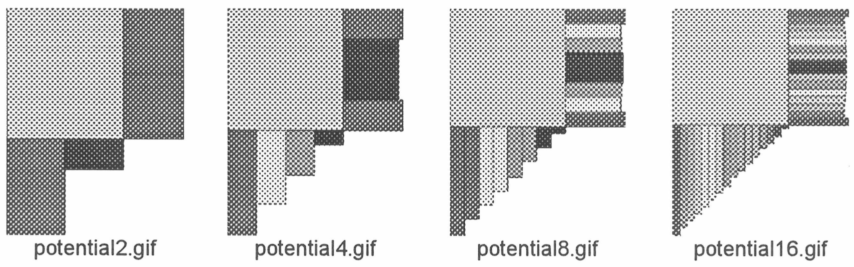
The *HTML* code to allow the input of such data would be :

```
<HTML>
<HEAD><TITLE>POTENTIAL</TITLE></HEAD>
<BODY>
<H1>POTENTIAL -Heat Flow Example</H1>
<P><IMG SRC="potential.gif" WIDTH=227 HEIGHT=179></P>
<FORM ACTION="http://www.cesec.ufpr.br/cgi-bin/potential.exe">
<P>1. Potential on the left side :</P>
<INPUT TYPE="text" VALUE="100" SIZE=8 NAME="leftpotential">
<P>2. Domain side :</P>
<INPUT TYPE="text" VALUE="2" SIZE=8 NAME="domainside">
<P>3. Number of boundary elements in each face :</P>
<INPUT TYPE="radio" NAME="numelements" VALUE="2"> 2
<INPUT TYPE="radio" NAME="numelements" VALUE="4" CHECKED> 4
<INPUT TYPE="radio" NAME="numelements" VALUE="8"> 8
<INPUT TYPE="radio" NAME="numelements" VALUE="16"> 16
<P><INPUT TYPE="submit" VALUE="EXECUTE"></P>
</FORM>
</BODY>
</HTML>
```

In the above code, an image file is used to illustrate the definition of the problem. The side of the square domain and the temperature of the left side are given in single-line text boxes already known. However, the number of elements of each side is defined with a control known as radio-button. It allows to the user to choose one among some predefined options with the use a set of `<INPUT TYPE="radio">` *tags*. Only the checked option is sent on the data string.

The Potential application can still be improved using the *HTML* capability of presenting images to provide a graphic output to the program. Once the options to the number of elements in each side of the square are predefined in the *HTML* file, the user can create in advance the same number of image files to illustrate each situation.

FIG.11 THE POTENTIAL APPLICATION GIF FILES



A *CGI* output consists of an *HTML* file and as an *HTML* file can reference an image file, a program can send back to the user the appropriate predefined image file, accordingly to the option chosen in the input process.

Consider the following code :

```
void Output(void) {
cout << "Content-type:text/html\n\n";
cout << "<HTML>\n";
cout << "<BODY>\n";
...
// output of numeric data
...
if (N == 8)
cout << "<P><IMG SRC=potential2.gif></P>\n";
else if (N == 16)
cout << "<P><IMG SRC=potential4.gif></P>\n";
else if (N == 32)
cout << "<P><IMG SRC=potential8.gif></P>\n";
else if (N == 64)
cout << "<P><IMG SRC=potential16.gif></P>\n";
cout << "</BODY>";
cout << "</HTML>";
}
```

In the code above, the output results are sent to the user followed of an image chosen accordingly the number of elements of the problem being analyzed.

If the graphic precision is not the main issue, this little trick can be used to enhance the application. The images are used only to give an approximated idea about the behavior of the heat flow when the number of elements varies. This approach cannot be used when an accurate precision is needed by the program.

5.5 USING JAVA APPLETS

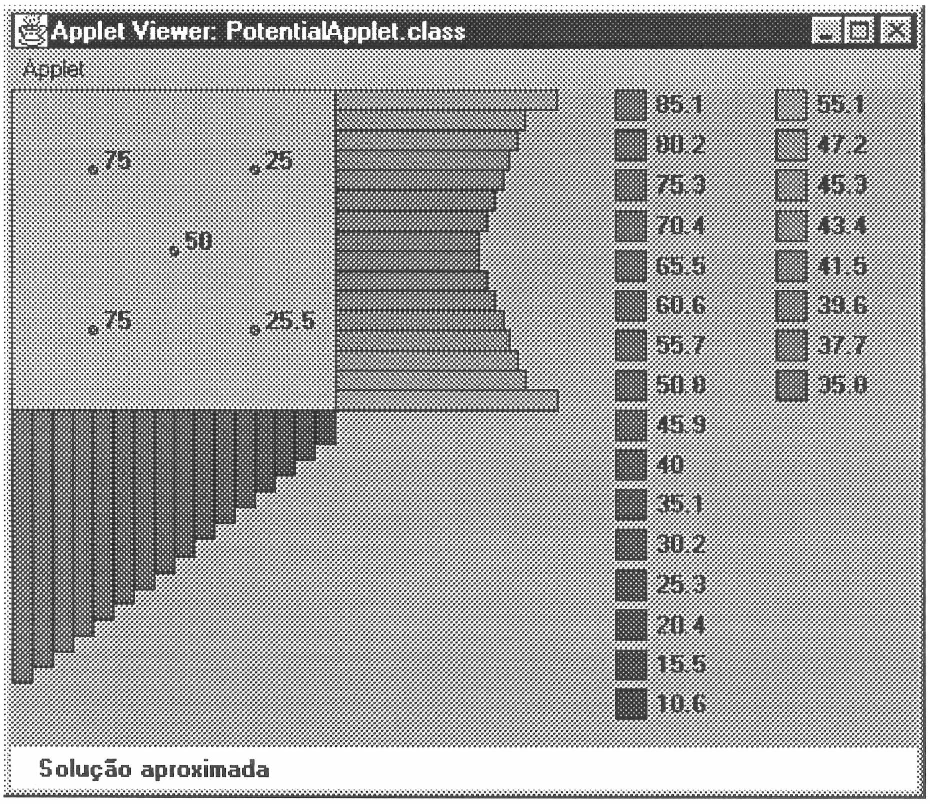
The use of predefined image files to provide graphic output is a very simple technique and, under some aspects, it can be considered very poor and limited. The Potential application presented in the previous section could take advantage of the use of a Java *applet* to present a better graphic output to the user.

As stated in the Chapter 4, a Java *applet* is a small application that is accessed on an Internet *server*, transported over the network, automatically installed and run as part of a Web document.

A Java *applet* supports graphics shapes that can be drawn edge-only or filled in a selected color. Multiple type fonts and controls are also supported. Parameters can be passed to a Java *applet* with the use of *HTML tags*.

A better approach to the Potential application would send a Java *applet* to the *client* web browser and pass to it the parameters needed to draw the output, as shown below :

FIG.12 THE POTENTIAL APPLET



In this case, the *HTML tags* and the variables containing the values used to draw the graphic output must be combined within the output function. This approach may not be a good choice when transferring a large amount of data to the Java *applet*.

Another good opportunity to use a Java *applet* in the Potential application is in the input *HTML* form. The use of an *applet* allows the user to have a graphic definition of the problem before the data is sent to the server. This *client*-side visualization associated with a pre-verification of data can reduce the processing load on the *server* side. Once the *Java applets* have the response capability to mouse and keyboard events, an interactive interface can be create

5.6 DEVELOPMENT OF WINDOWS CLIENT SIDE APPLICATIONS

The *Common Gateway Interface (CGI)* is the standard to create *server*-side applications. Many scientific and engineering applications are fitted to act in the server side and used with a Web browser. However, even with the use of Java *applets*, some situations demand a more powerful interface for data input or graphic visualization.

This section presents the use of two Internet *protocols* to develop *client* applications for the Windows operating system. As any standard Windows application, an Internet *client* program can benefit itself of all available resources of the environment like database access and 3D graphic visualization and animation. The *HTTP protocol* can be embedded in a Windows application, giving it the capability to start a *CGI* program, pass data to it and receive the results from the server. The *FTP protocol* is best suited for file-transfer applications. In some situations, data files must be shared by a group of users distributed in different locations and the server is used to hold these files. Using the *FTP protocol*, the application can access a file placed in a Web *server* in the same way it access a file located in the local hard disk.

5.6.1 THE WINDOWS INTERNET API (WININET)

Windows Internet applications can be developed using one of two options : the Winsock or the WinInet APIs. *WinInet* is a high-level interface to the *HTTP* and *FTP protocols*. It used to add communication capabilities to a Window application. Complex tasks become simpler with this high-level programming interface.

To demonstrate the use of WinInet API in the creation of client applications, a Windows program was developed to access text files in an *server* using the *FTP* Internet *protocol* and provide an graphical output of the data embedded within those files. To accomplish it, those applications used the OpenGL[5,29] API that allows graphic rendering in a whole set of computers and operating system.

FIG.13 A GRAPHIC VISUALIZATION CLIENT APPLICATION

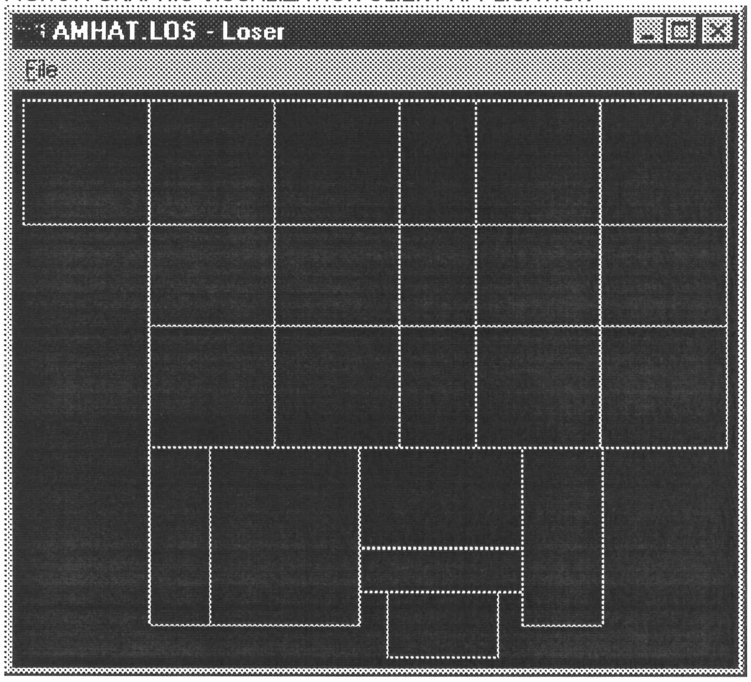
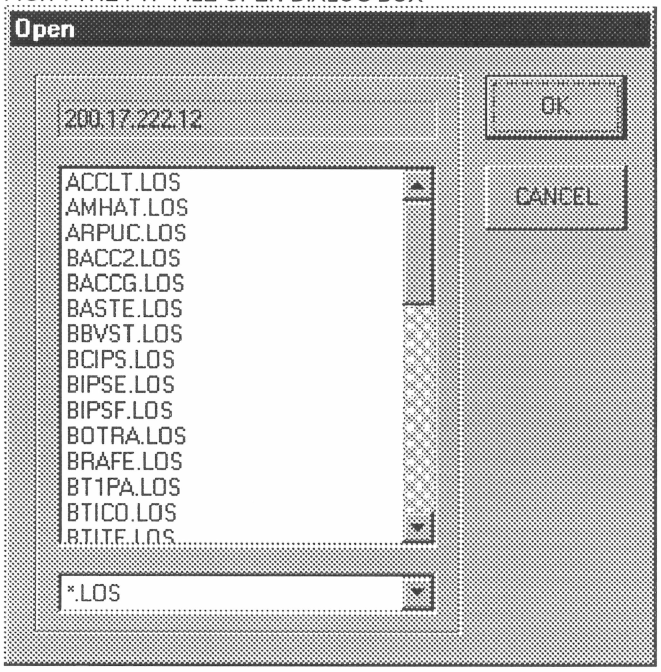


FIG.14 THE FTP FILE OPEN DIALOG BOX



200.17.222.12

ACCLT.LOS
AMHAT.LOS
ARPUC.LOS
BACC2.LOS
BACCG.LOS
BASTE.LOS
BBVST.LOS
BCIPS.LOS
BIPSE.LOS
BIPSF.LOS
BOTRA.LOS
BRAFE.LOS
BT1PA.LOS
BTICO.LOS
BTITE.LOS

*.LOS

Although the information is used by that program only to give a graphic visualization, it could be easily improved to become preprocessors or postprocessor to existing engineering applications. The main goal of the application is to provide a good knowledge of how applications can take full advantage of Windows resources and, at the same time, to make good use of the Internet capabilities.

6. CONCLUSIONS

Most of the students and researchers involved in scientific and engineering applications development are dealing with the Windows based applications. The knowledge of two primary tools is needed to develop a Windows application. The first one is a computer language, which is used to build the basic structure of the application. The second one is the Windows Application Programming Interface (API). The Windows API is a set of more than 2500 functions and its understanding demands experience and a large amount of time.[23,24]

Science and engineering students and researchers can seldom spend theirs efforts in learning a complex environment like Windows with the only purpose of producing a user interface to the application. Instead, using a better approach, the programmer can focus on the problem being studied and make a good and efficient use of his time and skills. Another key point about academic research is that its result may or should benefit the community. Using the right environment, the knowledge can be disseminated with minimum effort and cost.

In this work, the Internet is proposed as the environment to the development of scientific and engineering applications. Once the *HTML* specification has most of the standard and well-known controls provided by the Windows interface, a programmer can provide almost the same functionality with the knowledge of only a few *HTML tags*. The data input and output become a simpler task that does not demand advanced programming techniques.

Although the main reason for using of Object Oriented Programming is the increase of the complexity of applications, a practical problem faces the programmer. The dissemination of the *visual* environments had turned the development of Windows applications an easy task. However, the code of the applications created with those tools use Oriented Object Programming and the knowledge of this paradigm cannot be avoided. One example of Object Oriented Programming API is the Microsoft Foundation Classes (MFC)[19], that replaces the original Windows API. Another example is the Java programming language that also demands a good knowledge of Object Oriented Programming.

By using the Object Oriented Programming paradigm, applications of any level of complexity can be created. Despite the problem being studied, the use of this programming technique allows the development of a set of related applications with a standard approach. The code implementation can be done in a very efficient way. The program maintenance or improvement can also become easier tasks. The final remark about Object Oriented Programming is that this paradigm is fitted to be used by professional programmers as well as by less-experienced ones.

6.1 RECOMMENDATIONS AND DIRECTIONS FOR FUTURE WORKS

The future seems to point to an increase of Internet capabilities and applications. Internet is the environment in which all recent efforts of the computer industry are focused. Hardware and software together have been directed to the network capabilities provided by the Internet. Commercial sites and applications are coming to be landmarks in the business community. Daily routines have been changing with the necessity of adaptation to the new media. Academic research is intended to be ahead of the knowledge development and this is the ultimate reason for adopting this environment for the development of scientific and engineering applications.

Future developments and research can extend this work in many directions.

The advanced features of Object Oriented Programming can be used to enhance the Potential application, shown in Chapter 5 of this text. By using pointers to objects and the generic classes presented in Chapter 3, the application could give the choice between constant and linear elements. In this way, the program allows increasing the number of elements or to use a different interpolation function. Academic ones are some of the best for the application of the concepts presented in this work.

The Potential and Elastostatics codes can also be improved by giving them a graphical and interactive interface. To accomplish this the Java language must be studied in detail to a better understanding of its capabilities and limitations.

In the context of a numerical methods course, a set of applications to solve framed structures as well as plane strain, plane stress and plate problems could be easily implemented.

By using Object Oriented Programming, applications can be created or enhanced to support advanced analysis such as dynamic behavior, nonlinearities, plasticity and many others. This paradigm allows the implementation of a large project growing accordingly to the availability of resources.

Internet resources are also another direction to research. There are a great number of products not mentioned in this work. A special care must to be taken with two main issues : portability and standardization. Once the Internet gathers a variety of computer manufacturers, operating systems and software developers, the portability of the code and of the programmer is the most desirable feature of any product in study. Many resources are claimed to be standard by some recognized boards or by major hardware or software manufacturers. However, the future of any product or resource depends mainly in its accepting by the community and the market.

This work recommends the embedding of the Internet capabilities into any engineering program. As a result of the computational mechanics research, the applications developed here aimed the solution of engineering problems. However, the development of distributed applications can benefit the project area but can also be used by the field engineers. In the field of civil engineering, the status of a building project[35], information from the construction site[38], or just project plans can be shared through the Internet by engineers and contractors, workers and owners.[42,49]

This work attempts to be a first step to students and researchers in the development of engineering applications to the Internet. It is a simple and practical tutorial to present basic concepts about a programming technique and a network environment. Much work has to be done and countless are the options for the use of the technology presented here.

Written by a programmer, this dissertation is addressed to programmers. Some previous experience and knowledge made this work. Recent knowledge and research were also needed and will be needed in its sequence. For the future, the final recommendation or direction is to read and to try. Any path chosen will bring good results with the combination of imagination and much work.

"The science is on the books, not in the people's mind !"

Mildred Ballin Hecke, D.Sc.

REFERENCES

1. BREBBIA, Carlos Alberto; DOMINGUEZ. J. **Boundary elements** : an introductory course. 2. ed. Computational Mechanics, 1992.
2. CHAPMAN, Davis. **Building Internet applications with Delphi 2**. QUE, 1996.
3. FELTON, Mark. **CGI Internet Programming** : with C++ and C. Prentice-Hall, 1997.
4. FOLEY, James D. et al. **Computer graphics** : principles and practice. 2. ed. Addison-Wesley, 1997.
5. FOSNER, Ron. **OpenGL programming for Windows 95 and Windows NT**. Addison-Wesley, 1996.
6. GUNDAVARAM, Shishir. **CGI Scripting on the World Wide Web**. O'Reilly & Associates. 1996.
7. HEINY, Loren. **Advanced graphics programming using C/C++**. John Wiley, 1993.
8. HEINY, Loren. **Windows graphics programming with Borland C++**. 2. ed. John Wiley, 1994.
9. HUNT, Craig Estabrook. **TCP/IP Network Administration**. 2. ed. O'Reilly & Associates, 1998.
10. LEAVENS, Alex. **Designing GUI applications for Windows**. M&T Books, 1994.
11. LEVINE, John. **Programming for graphics files in C and C++**. New York : John Wiley, 1994.
12. MEYER, Bertrand. **Object oriented software construction**. 2. ed. Prentice Hall, 1997.
13. MYERS, Roy E. **Microcomputer graphics**. Addison-Wesley, 1982.
14. MUSCIANO, Chuck; KENNEDY, Bill. **HTML** : the definitive guide. O'Reilly & Associates, 1998.
15. NAUGTHON, Patrick; SCHILDT, Herbert. **Java** : the complete reference. McGraw-Hill, 1997.
16. ROBERTS, Dave. **Developing for the Internet with Winsock**. Coriolis, 1995.
17. RUMBAUGH, James et al. **Object oriented modeling and design**. Prentice-Hall, 1991.
18. SCHILDT, Herbert. **Advanced Windows 95 programming in C and C++**. McGraw-Hill, 1996.
19. SCHILDT, Herbert. **MFC programming from the ground up**. McGraw-Hill, 1996.
20. SCHILDT, Herbert. **Turbo C/C++** : the complete reference. 2. ed. McGraw-Hill, 1992.
21. SCHILDT, Herbert. **Windows 95 programming in C and C++**. McGraw-Hill, 1995.
22. SCHILDT, Herbert. **Windows 95 programming nuts & bolts** : for experienced programmers. McGraw-Hill, 1995.
23. SIMON, Richard. **Windows 95 common controls & messages API Bible**. Waite, 1996.
24. SIMON, Richard. **Windows 95 WIN32 programming API Bible**. Waite, 1996.

25. SKONNARD, Aaron. **Essential WinInet**. Addison Wesley, 1999.

26. STROUTUP, Bjarne. **The C++ programming language**. 3. ed. Addison-Wesley, 1997.

27. **READINGS on Microsoft Windows and Wosa**. Microsoft Press, 1995.

28. WALL, L.; SCHWARTZ, R. L. **Programming Perl**. O'Reilly & Associates, 1991.

29. WRIGHT JR, Richard S.; SWEET, Michael. **OpenGL superbible**. Waite, 1996.

30. FAVELA, **Jesus. An object oriented approach to the design of CAE systems**. Boston, 1989. Dissertation (Master of Science) – Department of Civil Engineering, Massachusetts Institute of Technology.

31. GUIMARÃES, Luiz Gil Solon. **Disciplina de programação orientada a objetos para análise e visualização bidimensional de modelos de elementos finitos**. Rio de Janeiro, 1992. Dissertação (Mestre em Ciências) – Departamento de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro.

32. WORLD CONGRESS ON COMPUTATIONAL MECHANICS (4 : 1998 : Buenos Aires). **Abstracts**. IACM, 1988.

33. AKIN, J. E. Object oriented programming via Fortran90. In: Engineering Computations. MCB, v. 16, n. 1, p. 26-48, 1999.

34. BAILEY, Simon F.; SMITH Ian F. C. Case-based preliminary building design. In: **Journal of Computing in Civil Engineering**. ASCE, v. 11, n. 4, p. 454-468,oct. 1994.

35. DI FELICE, Paolino. Why engineering software is not reusable: empirical data from an experiment. In: **Advances in Engineering Software**. Elsevier, v. 29, n. 2, p. 151-163, mar. 1998.

36. FEIJO', Bruno; BENTO, J. A logic-based environment for reactive agents in intelligent CAD systems. In: **Advances in Engineering Software**. Elsevier, v. 29, n. 10, p. 825-832, dec. 1998.

37. FRUCHTER, Renate et al. Interdisciplinary communication medium for collaborative conceptual building design. In: **Advances in Engineering Software**. Elsevier, v. 25, n. 2/3, p. 89-101, mar./apr. 1996.

38. ISREB, M.; KHAN, A. I.; PARKER, B. A. Adaptative finite element mesh refinement: a call module on the WWW. In: **Computer and Structures**. Pergamon, v. 65, n. 2, p. 169-175, 1997.

39. JU, Jianing; HOSAIN, M. U. Finite-element graphic objects in C++. In: **Journal of Computing in Civil Engineering**. ASCE, v. 10, n. 3, p. 258-260, jul. 1996

40. KHEDRO, Taha. A distributed problem-solving approach to collaborative facility engineering. In: **Advances in Engineering Software**. Elsevier, v. 25, n. 2/3, p. 243-252, mar./apr.1996.

41. KIWAN, M. S.; MUNNS, A. K. A neutral object data model for integrated building design and construction environment. In: **Advances in Engineering Software**. Elsevier, v. 25, n. 2/3, p. 131-140, mar./apr.1996.

42. LAW, Kincho H.; BARSALOU, Thierry; WIEDERHOLD. Management of complex structural objects in a relational framework. In: **Engineering with Computers**. Springer-Verlag, v. 6, n. 2, p. 81-92, spring 1990.

43. MACKIE, R. I. An object oriented approach to fully interactive finite element software. In: **Advances in Engineering Software**. Elsevier, v. 29, n. 2, p. 139-149, mar. 1998.

44. MACKIE, R. I. Using objects to handle complexity in Finite Element Software. In: **Engineering with computers**. Springer-Verlag, v.13, n. 2, p. 99-111, 1997.

45. MONI, Sheloney; WHITE, Donald W. Frameview: object-oriented visualization for frame analysis. In: **Journal of Computing in Civil Engineering**. ASCE, v. 10, n. 4, p. 276-285, oct. 1996

46. REMY, Philippe; DEVLOO, Bernard; ALVES FILHO, José Sergio Rodrigues. An object oriented approach to finite element programming (phase I): a system independent windowing environment for developing interactive scientific programs. In: **Advances in Engineering Software**. Elsevier, v. 14, n. 1, p. 41-46, 1992.

47. RETIK, A.; KUMAR, B. Computer-aided integration of multidisciplinary design information. In: **Advances in Engineering Software**. Elsevier, v. 25, n. 2/3, p. 111-122, 1996.

48. SUNIL, K. Evt; KHAYYAL. Sari; SANVIDO, Victor E. Representing building product information using hypermedia. In: **Journal of Computing in Civil Engineering**. ASCE, v. 6, n. 1, p. 3-18, jan. 1992.

49. ZIGA, Turk; ISAKOVIC, Tatjana; FISCHINGER, Matej. Object-oriented modeling of design system for RC buildings. In: **Journal of Computing in Civil Engineering**. ASCE, v. 8, n. 4, oct. 1994.

50. HARDWICK, Martin; SPOONER, David L. Data protocols for the industrial virtual enterprise. In: **IEEE Internet Computing Online**. <http://computer.org/internet/9701/hardwicki9701.html>, in 09.06.1997.

51. REGLI, William C. Internet-enabled computer-aided design. In: **IEEE Internet Computing Online**. <http://computer.org/internet/9701/regli9701.html>, in 27.03.1997.

52. SCHEER, Sergio; POMPEU, Renato Cesar. **Virtual environments for the engineering teaching and learning**. CONGRESSO IBERO-LATINO AMERICANO DE MÉTODOS COMPUTATIONAIS EM ENGENHARIA - CILAMCE (XX : 1999 : São Paulo). 3-5 nov. 1999.

53. **HTTP – Hypertext Transfer Protocol**. <www.w3.org/Protocols> In : 18.10.1999.

54. SCHWABE, Daniel; ROSSI, Gustavo. **The Object-Oriented Hypermedia Design Model (OOHDM)**. <www.inf.puc-rio.br>. In 15.10.1999

UFPR

MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE CIÊNCIAS EXATAS - Departamento de Matemática
SETOR DE TECNOLOGIA - Departamento de Construção Civil
COORDENAÇÃO DO CURSO DE PÓS-GRADUAÇÃO EM MÉTODOS NUMÉRICOS EM ENGENHARIA

RELATÓRIO DA DEFESA DE DISSERTAÇÃO DE MESTRADO

Aos 04 dias do mês de outubro de 1999, no Auditório do SIMEPAR, Universidade Federal do Paraná, foi instalada pelo Professor Waldyr de Lima e Silva Júnior, Coordenador do ppgMNE - Programa de Pós-Graduação em Métodos Numéricos em Engenharia, a Banca Examinadora para a décima primeira Dissertação de Mestrado em Métodos Numéricos em Engenharia, Área de Concentração em Mecânica Computacional. Estiveram presentes ao Ato, além do Coordenador do Programa de Pós-Graduação em Métodos Numéricos em Engenharia, professores, alunos e visitantes.

A banca examinadora, atendendo determinação do Colegiado do Programa de Pós-Graduação em Métodos Numéricos em Engenharia, ficou constituída pelos professores Bruno Feijó, Dr.,Departamento de Informática, da Pontifícia Universidade Católica do Rio de Janeiro; Waldyr de Lima e Silva Júnior, Ph.D., co-orientador, Centro de Estudos de Engenharia Civil Universidade Federal do Paraná; Ricardo Mendes Junior, D.Eng., Centro de Estudos de Engenharia Civil, da Universidade Federal do Paraná e Sérgio Scheer, D.Sc., Centro de Estudos de Engenharia Civil, Universidade Federal do Paraná, orientador principal, a quem coube a presidência dos trabalhos.

Às quatorze horas, a banca iniciou seus trabalhos, convidando o candidato Manoel Theodoro Fagundes Cunha a fazer a apresentação do tema da dissertação intitulada "A Practical Approach in the Development of Engineering Applications for the Internet Using Object Oriented Programming". Encerrada a apresentação, iniciou-se a fase de argüição pelos membros participantes.

Após a argüição, a banca reuniu-se para apreciação do desempenho do pós-graduando e definição de notas.

A banca considerou que o pós-graduando fez uma apresentação com a necessária concisão e que esclareceu os pontos necessários ao melhor entendimento da dissertação, respondendo as questões formuladas.

A Dissertação apresenta contribuição à área de estudos e não foram registrados problemas de estrutura e redação, resultando em plena e satisfatória compreensão dos objetivos pretendidos.

Tendo em vista a dissertação e a argüição, a banca atribuiu as seguintes notas: Prof. Waldyr de Lima e Silva Júnior, nota 10,0 (dez); Prof. Bruno Feijó, nota 10,0 (dez), Prof. Ricardo Mendes Junior, nota 10,0 (dez) e Prof. Sergio Scheer, nota 10,0 (dez).

A média obtida 10,0 (dez), resulta na **aprovação** do candidato, (de acordo com determinação dos Artigos 32, 33 e 34 da Resolução 74/94-CEP de 21/10/94), e corresponde ao conceito " A ".

Curitiba, 04 de outubro de 1999

[illegible]

Prof. Sergio Scheer, D.Sc.
Presidente

[illegible]

Prof. Waldyr de Lima e Silva Júnior, Ph.D.

[illegible]

Prof. Bruno Feijó, Ph.D.

[illegible]

Prof. Ricardo Mendes Junior, D.Eng.