소프트웨어 재공학과 유지보수 지원을 위한 툴의 개발

김 행 곤[†] 황 선 명[††]

요 약

소프트웨어 재공학 툴은 소프트웨어 유지보수 생산성과 그작업의 품질 향상에 기여한다. 재공학은 프로그램의 구성요소에 대한 기능 변경이 아닌 행위 변경(객체의 이름과 정의, 재구성프로세스 논리)을 일반적으로 관리한다. 본 논문은 객체 지향 프로그램언어(C++)로 작성된 원시 코드의 유지 보수를 위한 소프트웨어 재공학 툴인 InMaC++(Interactive Maintenance for C++)의 개발에 대해 서술한다. InMaC++는 원시 코드와 다이어그램(설계정보)형태 모두를 표현할 수 있고 편집 기능이 가능하며 원시 코드에서 다이어그램으로, 다이어그램에서 원시 코드로의 변형이 가능하다. 따라서 이들 변형을 통해 기존 코드의 유지보수와 재공학이 가능하다. 특히 시스템의 검색기능은 그래픽 인터페이스로 구현되었으며 InMaC++는 객체 지향 프로그램의 메소드, 속성, 클래스등의 구성 요소를 기반한 데이터베이스를 포함하고 있으며, 이 모델은 구현과 사용의 편리성과 툴의 구성을 용이하게 하기 위해 4가지의 객체 클래스와 3가지 관계를 정의하고 이들 객체 클래스의 검색을 위해 시스템 제공 단순 질의어가 제공된다.

Development of the Tool for Software Re-engineering and Maintenance

Haeng-Kon Kim[†] and Sun-Myung Hwang[††]

ABSTRACT

Re-engineering tools can substantially increase software maintenance productivity and the quality of maintenance work. Re-engineering usually involves changing the form(e.g.changining objects names and definitions, restructuring process logic) of a program. In this paper, we describe the design and implementation of InMaC++ that is a software tool oriented towards maintenance of C++ object oriented programs. With InMaC++, programms can be displayed and edited in two forms : as the code and as the diagram InMaC++ also contains transformations in both directions, i.e. from code to diagram and from diagram to skeletons of code. Hence, it is suitable for re-engineering and maintenance of existing code. Specially designed browsers implement the graphical interface. InMaC++ contains a database that is based on a simple but effective data model of C++ programs. The model contains only four object classes and three relations, which makes the tool small, and easy to implement and use. A simple query language allows browsing through the database.

1. Introduction

Software development is an evolving process resulting in larger and larger programs. As programs grow in size, they become more complex and harder to maintain. Futhermore, larger programs require more intensive maintenance. This fact could be attributed to many reasons, among them corrections of errors, changes in

· This paper was supported by NON DIRECTED RESEARCH FUND, Korea Research Foundation, 1994
† 정 회 원 : 효성여대 전자계산학과 교수
†† 정 회 원 : 대전대학교 전자계산학과 교수
논문접수 : 1994년 5월 7일, 심사완료 : 1994년 7월 25일

requirements, enhancements demanded by the users, efficiency improvements, changes in hardware or software configurations, etc. Thus it is not a surprise to see that the maintenance of software is the most expensive part of the software lifecycle. Hence it makes good economic sense to try to automate the activities of software maintenance and create tools that would be a help in this area [1].

In order to modify a program, software maintainers have to understand the program. Understanding programs is one of the most time consuming activities of software maintenance. This partially due to the fact that quite often the only reliable and available documentation is the source code of the program itself, and all relevant information must be extracted from it. But programs are complex, abstract objects. They include many components with many different attributes that are interrelated in complicated ways. This makes it difficult for programmer to understand and navigate through complex interrelationships among pro-gram parts. Moreover, textual representation of a program does not reveal such information right away. In the text of the code, important partitions and relations (i.e. program architecture, or programming-in-the-large) are scattered in large amounts of local and trivial information(i.e programming-in-the-small). To modify a program successfully, program architecture must be well understood by the programmer and correctly used.

One way to alleviate this problem is to create a tool that allows the programmer to deal with program architecture directly, without having to extract it repeatly and and manually from the code. Another improvement is to represent the architecture graphically and in this way to make it more understandable and easy to deal with. Such a tool maintains the graphical representation of the program and provides the programmer with visual editor to build and modify the program. InMaC-- is a tool aimed at providing such support.

2. Related works and high level system understanding

Informations processed in computers are so various that information presentation problems have been studied in domain dependent ways. There have been a lot of studies on the visulaization of specific information. Recently many techniques have been studied to increase the understandability through visualization such as automated diagram, hypertext and documentations. All of these techniques are not related object-oriented software systems but traditional systems.

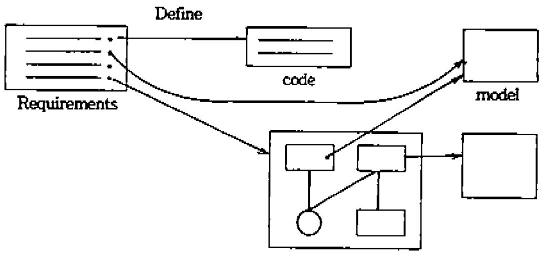
2.1 C information abstractor

C information abstractor[2] decides the detail implementation information with lower-level abstract module, especially if the user defines the C subprogram and module. It is used a detail high-level representation of interface by using well defined conceptual model and library information system. An automatic retrival system and representation systems are subunit of C abstractor. Unfortunately, we can only use the C information abstractor for traditional software component especially for C rather than object-oriented software component for C++, and dynamic behaviors for object.

2.2 Plain text

Plain text system[3] is the general understanding system that consists of network with text, diagram and source code. It uses the hypertext techniques. The nodes with information is usually viewed by users. Connection between nodes can be drawn by arrows and contain the several design information such as node explaination, design diagrams and expression of design step. Given original textual code can be expressed by hypertext and understood without changing the original file. An examples for Plain text is shown in (Fig. 1). User for component can access the several simple information and can have the flexible user interface.

However, plain text can not support the information hiding and inheritance, because it does not use the object-oriented paradigm. In addition, it can not filter the requirement, design and source code information as low level abstractor.



(Fig. 1) Plain text

2.3 AIPS [4]

An information presentation system is a knowledge based information presentation system implemented as a KLONE taxonomic hierarchy of display structure descrptions. A system for construction user interface display is built on knowledge representation systems. In order to realize a general presentation system which covers a wide range of presentation variations. It is important to build a knowledge based of presentation information like these systems.

High level system understanding is chiefly needed when a maintainer is coming to grips with a system for the first time. The maintainer needs some way to sort out the components and perceive the overall architecture of the system. A high level understanding will give a maintainer a framwork to help make sence of the more detailed information acquired as specific maintenence tasks are undertaken[5].

There is little tool support for high-level system understanding even for conventional (i.e.,nonobject— oriented)systems. Several researchers have suggested using clustering methods of different kinds to identify system structure. However, we know of no generally available tools implementing these ideas.

The module calling hierarchy or structure chart can be generated by several existing tools. Calling hierarchies are a useful tool for understanding system designed using funtional decomposition approaches in which the main packaging unit is the processing module (e.g., a function or procedure). In such system the top level "main module" will likely be a good place to start in understanding and, if the modules subordinate to it are resonably cohesive, examining them may give a quick overview of system functions.

But in OOP, the calling hierarchy would be a hierarchy of methods, which has several disadvantages. First, the dynamic binding problem may make the hierarchy difficult to compute. Second, there may be no real "main" method in the system. This is one of the facts about object oriented design that beginners tend to find disconcerting. Finally, a hierarchy of methods loses sight of the grouping of methods in objects, which is presumably the most important aspect of the design. An obvious understanding aid would be the object class hierarchy, but because it groups objects with similar methods, it fails to show how the objects combine to provide the different functional capabilities of the program. For example, in Smalltalk/V the Pane object work together with dispatcher objects to provide wimdows for text editing. But these objects are in different class hierarchies.

One possible high-level understanding tool would be a display of the diagram of the "Class uses class" dependency. However the result will be a diagram not a hierarchy.

Graph are notoriously more difficult to display and comprehend than trees. In a medium scale system with hundreds of object classes the diagram may not be particularly comprehensible.

Environments for object-oriented maintenance should probably provide several alternative clustering methods that can be chosen by the user during system exploration. The "Class Uses Class" dependency could provide the links for clustering but some experimentation will be needed to find the most useful methodologies.

3. Dependences in C++

A dependency in a software system is, informally , a direct relationship between two entities in the system X→Y such that a programmer modifying X must be concerned about possible side effects in Y. Earlier reports [6] analyzed the dependencies in conventional software systems. The main kinds of entity considered were data items (or variable), processing modules, and data types. Dependencies were classified as follows:

① data dependencies between two variables
② calling dependencies between two modules
③ functional dependencies between a module and the variables it computes
④ definitional dependencies between a variable and its type.

To deal with C++ languages, we need to add the following kinds of entities

① object classes
② methods(which are specific code segments)
③ messages(which may be thought of as "names" of methods).

Variables may now represent instances of an object class instead of, or in addition to, conventional data values. Object classes may be thought of as special kinds of types while methods are special kinds of processing module. However the use of polymorphism and hierarchy creates an explosion in the kinds of depecdencies that needs be considered:

① Class-to-Class Dependencies
- C1 is a direct super class of C2

- C1 is a direct sub class of C2
- C1 inherits from C2
- C1 uses C2 (which may be subclassified as "uses for interface", and "uses for implementation".)

② Class to Method
- method M returns object of class C
- C implements method M
- C inherits method M

③ Class to Message:
- C understands message

④ Class to Variable
- V is an instance of class C
- V is a class variable of C
- V is imported by M(ie, is a noniocal variable used in M)
- V is defined by M
- M refers to V

⑤ Method to Message
- message M+ is name of method M
- method M sends message M+
- method M sends message M+

⑥ Method to Method:
- method M1 invokes method M2
- Method M1 overrides M2

Environments for maintaining C++ need to provide ways of browsing these different kinds of relationships. The multidimensional nature of the interconnections will make it very difficults to use listing or text screen based systems for program understanding. Multi-window displays would seem to be a minimum requirement to display enough informations.

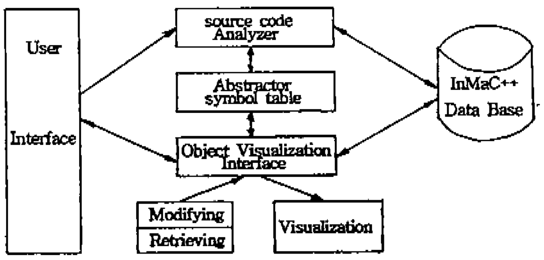
4. InMaC++ organization

4.1 System organization

The input data for InMaC++ is the C++ source program and the InMaC++ extract corresponding design informations after scanning code as follows.

① Class name
② Class attributes
③ Class internal methods
④ Other class definition
⑤ Client-server relation
⑥ Class Friendness
⑦ Class parameter to invoke message
⑧ Variable type returning message execution

(Fig. 2) shows the overall organization of InMaC++ system.



(Fig. 2) InMaC++ system Organization

4.2 General concepts

InMaC++ (Interactive Maintenance C++) is a language centered programming tool. InMaC++, a program can be represented in two different forms, the traditional one (code), or the graphical one. The distinguishing feature of InMaC++ is the data model that contains only four different classes of entities and three relations, yet it is able to support programming in a language as powerful and universal as C++.

The data model consists of the following entity classes:

① modules, i.e. files of source code (both compilation units and includefiles)

② declarations,which are divided into the following two subclasses,

(i) processes, i.e. main program, sub-routines, and funtions

(ii) commons, i.e. global data elements.

It also contains the following relations:

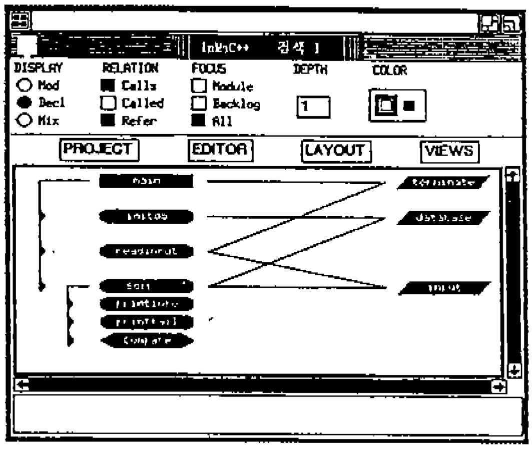
① The belong to relation that specifies whether any objects are parts of another object. In paticular, declarations belong to classes.

② The message interconnections processes. The processes and their messages constitute a message-diagram.

③ The reference relation interconnetions prosesses and commons. The define which processes have access to which commons.

Each declaration has attributes, the most important one being the name and the code of the delaration.

In the visual form, the program is represented by a diagram consisting of icons and lines between icons. (Fig. 3) shows the graphical representation of a small program for a compurterized telephone directory, where a person's telephone number is found



(Fig. 3) InMaC++ Organization

by searching a database. Notice that the diagram in (Fig. 3) is laid out two columns.

The left column consists of processes, and the right column consists of commons. The main program, subroutines, and funtions are represented by retangles, octagons and diamonds in the left column, repectively. The commons are represented by parallel in the right column. The arrows on the left represent the call relations among the processes, i.e. the call diagram. The lines in the middle represent the reference relations i. e. the reference diagram. We call this type of layout a two column graph (abbreviated 2CG).

The two-column graph is an original layout that was specifically developed for InMaC++. It is supported by an algorithm that automatically developed for in the columns. In the literature, several other layouts and layout algorithms were investigated. We chose the two-column layout because it is sufficient for our simple data model and it presents several advantages over the other layouts. In particular, it allow many icons and connecting lines without becomming unreadible. An even more important feature is the increment of two-column graphs. Whenever a small incremental change (such as addition or deletion of an icon) is made, the overall layout also changes only incrementally, even when allocated autum-atically by the algorithm. Hence it is easy for the user to find orientation in two-column graphs after edting changes, because icons are found in similar places.

The graphical interface is implemented by the browsers. Browsers are specially designed windows that support communication with the user. They allow the user to query the database, and to display the result of such queries graphically. Each brower is equipped with a query panel, pull-down cascading menus, a drawing area, scrollbars and a message area at the bottom (see the example in Fig. 3).

As programs increase in size, their correponding graphs become more complicated. Because of the large number of nodes and intersecting lines. In other to overcome this problem, we have implemented two features that make graphs more readable, views and diagram rearragement.

4.3 Views

The information about entities and relations in the programs is stored in the database. Views are subsets of the database that contain particular structural information of the program. The need for views is supported by the fact that programmers are usually not interested in the complete structure of a large software system. At any given time, they need to see some cross-section that contains the entities relevent to a particular task. Views are a general and effective way to overcome problems of tracing and navigating through a large database. Views are defined by queries which in turn are specified by the buttons of the query panel in the upper part of the browser.

In order to illustrate views, consider the view of (Fig. 3) that represents the declarations of a complete program. The program is displayed in the declaration mode, where only declarations are shown and modules are ignored. All declarations of the program are displayed (the button All is selected). Finally, the view contains call and reference relations (the buttons Call and Refer are selected), respectively.

(Fig. 4) contains a screen with browsers simultaneously displayed, where 검색 1 (BROWSER-1) displays the same program in the mixed mode(the button Mix is selected).

It displays not only declarations and their relations Call and Refer as in (Fig. 3), but also the modules (files) to which all these delclarations belong. 검색2(BROWSER-2) of

(Fig. 4) displays the same program in the module mode (the button Mod is selected). It shows all modules that the program consists of the call and reference relationships among them.

In general, queries are executed in the following way, first, all focus entities are retrieved. Then all additional entities which are transitively related to focus entities are retrieved, up to the depth specified in the query panel. The query consists of declaration mode (the button Decl is selected), the call relation (the button Calls is selected),and DEPTH is indicated as 2. The corresponding view contains the focus declaration main,all processes of depth 1 for call relation(i.e. all processes called by Main, in this case search, readinput and initdb), and all processes of depth 2 for call relation(i.e. all processes called by processes of depth 1, in this case compare, printinfo and printfail). The focus declaration main has been selected by the mouse as a part of the query. Similarly, views can be displayed using 'called by' (inverse of 'calls')and 'refer' relations. In the privious example, the focus of the query was a single declaration. However the focus can be selected in several different ways: all declarations of a module, all declarations of the backlog, or all entities of the program can be selected (the button module, Backlog or All selected), respectivity. The backlog is the collection of all the declarations that have been introduced in the database but have not been defined yet. This focus is particularly useful for top-down programming.

Another feature of a query is a colour. Colours are used in drawing both entities and relations of a given view. They are selected from a colour palette in the query panel.

Views can be either separated or appended to an existing view. Separate views display the result of a query in a newly created window. Appended views display the result of a query as a new part of a previously existing window, where the new view is superimposed on the old view. Appended views are powerful tools that help the user to navigate progressively through the database. Each of them can be displayed in a different colour, enhancing the under standability of the display.

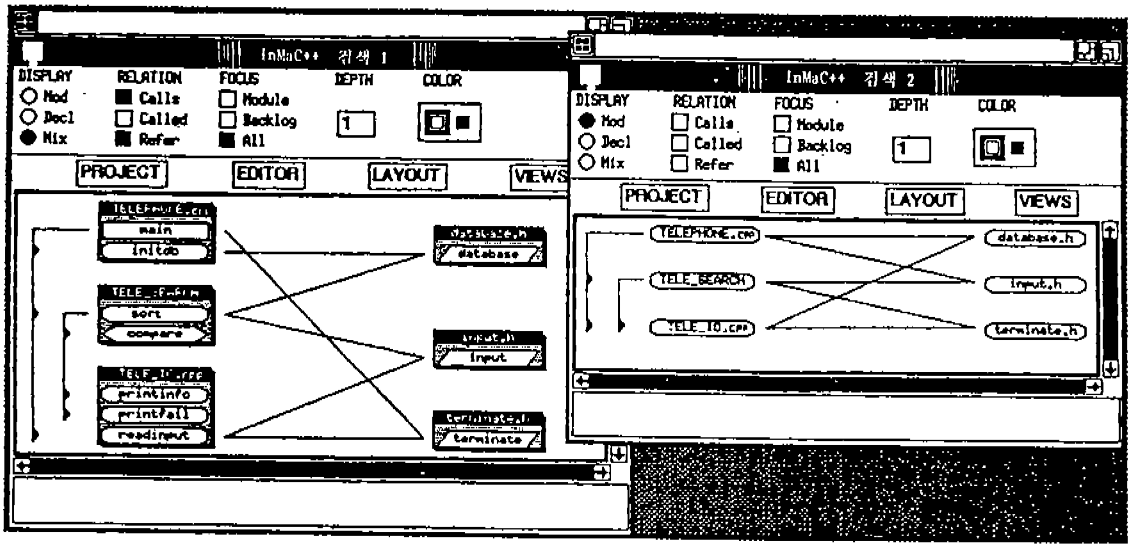
4.4 Graphic rearrangements

Despite the use of views, the displayed graph may still be complicated. In order to improve the readability, InMaC++ provides the following graph rearragement operations. The layout within a browser can be scaled down to make more entities and relations visible, or scaled up to provide a clearer view of the information displayed.

Icons can be moved to new locations within their column so that fewer crossings or shorter lines are achieved.

Hiding all relations entering or leaving an entity is a method of temporarily factoring out redundant information. The complementary operation, unhiding, can restore hidden relations.

Highlighting the relations and icon of an entity can make them more prominent and easier to follow. Highlighted lines are thicker and can be of a distinct colour.



(Fig. 4) InMaC++ Views

4.5 Editing operations

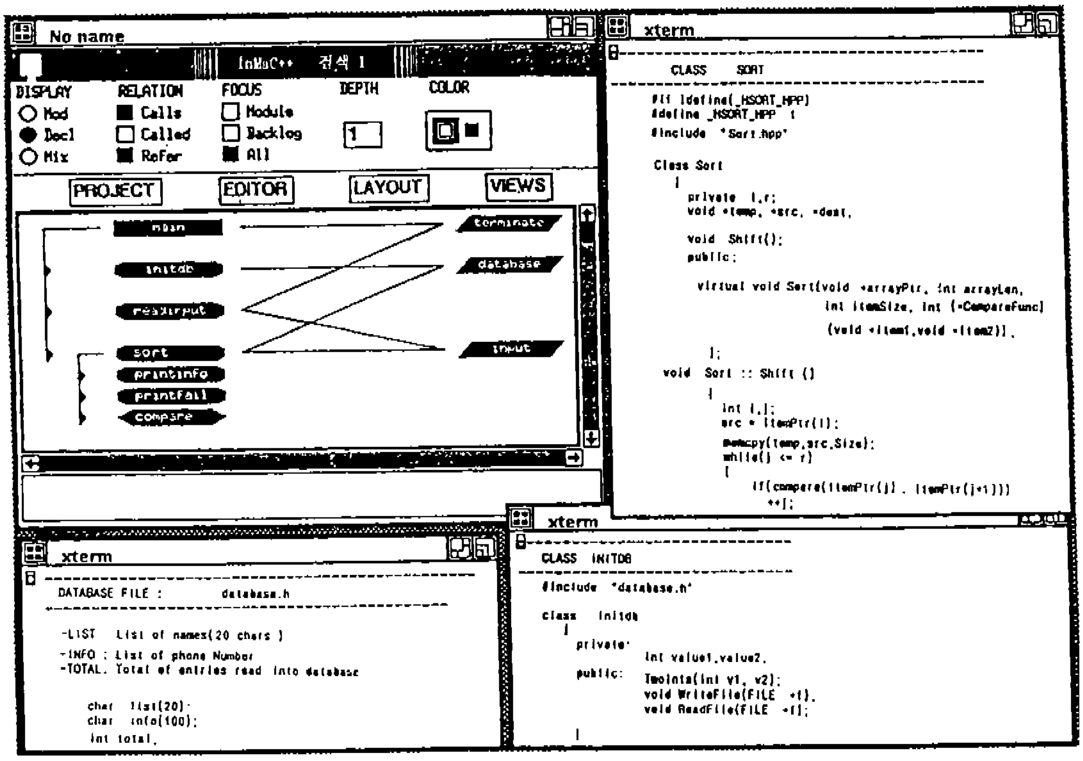
Editing operations are activities that chang the contents of the database. The operations of InMaC-+ are divided into four groups: diagram, diagram-to-code, code and code-to-diagram.

Graph operatins allow editing of the database content. They allow adding and deleting of entities and relations. On a more global scale, they allow loading and storing of the whole contents of a database.

Diagram-to-code operations generate skeletons of the code for every declaration of the diagram that does not have code. Such a skelton consists of all keywords and names that can be read from the diagram.

Code operations allow the user to edit source code through text-editing windows. InMaC++ uses several standard UNIX editors, that can be selected by the user.

Code-to-diagram operations function in the following way: the parser analyses the code and generates a symbol table with the semantic information for all entities. Then the analyser compares the information in the symbol table with that already stored in the database, and identifies all inconsistencies. If the symbol table contains additional information that does not conflict with the information already in the database, then the database is automatically updated. For example, all new entities in the code are automatically entered into the database. However if there is a conflict (multiple definitions of a declaration, etc.), the



DISPLAY: ( ) Mod [✓ Decl] ( ) Mix
RELATION: [✓ Calls] [ ] Called [✓ RoFer]
FOCUS: [ ] Module [ ] Backlog [✓ All]
DEPTH: 1

```
CLASS    SORT
--------------------------------------------
#if !define(_HSORT_HPP)
#define _HSORT_HPP  1
#include  "Sort.hpp"

Class Sort
   {
     private  l,r;
     void *temp, *src, *dest,

     void  Shift();
     public;

       virtual void Sort(void *arrayPtr, int arrayLen,
                         int itemSize, int (*CompareFunc]
                         (void *item1,void *item2)],
        ];
 void  Sort :: Shift ()
        {
          int i,j;
          src = itemPtr(l);
          memcpy(temp,src,Size);
          while(j <= r)
          [
               if(compare(itemPtr(j) , itemPtr(j+1)))
                 ++j;
```

```
DATABASE FILE :        database.h
--------------------------------------------
 -LIST   List of names(20 chars )
 -INFO : List of phone Number
 -TOTAL. Total of entries read into database

    char  list(20)·
    char  info(100);
    int total,
```

```
CLASS  INITDB
--------------------------------
#include  "database.h"

class   initdb
   {
     private·
              int value1,value2,
     public:  Twointa(int v1, v2);
              void WriteFile(FILE  *f),
              void ReadFile(FILE  *f];

     |
```

(Fig. 5) Graphs of the program in InMaC++

database is not updated and the user is notified.

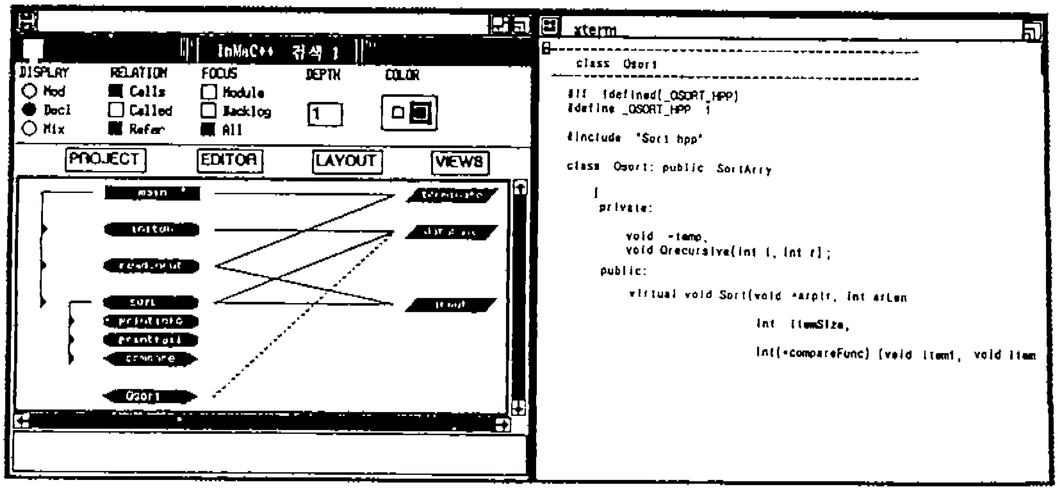
5. Examples of the use of InMaC++

InMaC++ supports maintenace, modifications and enhancements of already existing software. The relations stored in the database help the programmer to understand the code, and to follow the ripple effects of the modifications. This is illustrated by the modification of the phone directory program.

The phone directory program uses a file of names and phone numbers. The file is read by the program and an internal data structure is created. When the user enters the name of the person whose phone number is desired, the program searches the data structure and provides the answer. (Fig. 5) shows the complete diagram of the program, the bodies of the class initdb and the code of the include file database.h. The class initdb reads the file and enters the names and phone numbers into the data structure. The class readinput reads the name of the person whose phone number is requested into the string input, or the code for termination of the program.

The class sort uses a binary search method to locate the name in the database. The method printinfo and printfail are called by search to print the phone number or a message of unsuccessful search, respectively. The method compare perform alphabetical comparison on the input name with a name from the database.

The maintenance action on this program is the alteration of the algorithm of sort into a hashing search. As a first step, we create a new function called Quicksort with the graphical editor. Next, InMaC++ generates a skeleton of the code for Quicksort and rest of the code is entered by editing that skeleton. The code is parsed automatically and the reference relation from sort to database is added to InMaC++'s database. (Fig. 6) shows the new diagram with the function sort is highlighted and the reference sort to database indicated by a dotted line. The definition of sort is displayed in the right



```
class Qsort

#if !defined(_QSORT_HPP)
#define _QSORT_HPP 1

#include "Sort.hpp"

class Qsort: public SortArry

    {
     private:

         void  *temp,
         void Qrecursive(int l, int r);
     public:

          virtual void Sort(void *arptr, int arLen

                                  int  itemSize,

                                  int(*compareFunc) (void item1, void item
```

(Fig. 6) Graphics with the class Quicksort window. At this point, search is still using the old binary search algorithm, and sort is unconnected to the rest of the call diagram.

The definition of the database has to be modified in order to accommodate the new search method, i.e. the variable total will now hold the size of the hash table rather than the total number of records. Inspecting the diagram of (Fig. 6), notice that both initdb and sort are referencing database, therefore these are the processes affected by the change.

Their modifications are done via the text editor, and the code-to-diagram converter is invoked to update the information stored in InMaC++'s database.

6. InMaC++ evaluation

① object-oriented features

We evaluates the InMaC++ as information hiding encapsulation, inheritance and polymorphism as an object-oriented features. We compares it with interface abstractor and plain text as ⟨table 1⟩.

② user friendness

We also evaluates the InMaC++ in the view of user friendness as ⟨table 2⟩.

7. Future directions

There are several directions in which InMaC++ will be developed in the future. One

⟨Table 1⟩ object-oriented features in InMaC++

	InMaC++	Interface abstration	Plain text
Information hiding	0	0	x
Encapsulation	0	0	x
Inheritance	0	0	x
Polymorphism	0	x	x

⟨Table 2⟩ user friendness

	InMaC++	Interface abstration	Plain text
Direct manipulation	0	0	0
Mouce porting	0	0	0
Icon	0	X	X
Push botton	0	0	X
Zoom in/out	0	0	0
Help	0	X	X

of them is the link between C++ and the 'make' facility. There is a considerable overlap between the information stored in C++ database and information required by 'make'. The population of the database from 'make' files and the generation of skeletons of 'make' files from the database are on the drawing boards and will be incorperated into C++ in the future.

In reality, a whole family of tools for several languages can be developed. In fact, the most important limitation of this technology is the display, if the data model for the language can be displayed with the use of a two-column graph, then the language can be supported by a tool of the InMaC++ family. On the other hand, languages whose basic data model is more complex can not be supported by a tool of the InMaC++ family. For example, exceptions and exception handlers would create a problem for two column graphs, hence the language Ada, PL/1, etc. would be difficult to handle. Reserch is under way to develop a generator for the InMaC++ family of tools, where a specification language would describe the data model and the rest of the tool (with the exeption of the parser) would be generated from the specification.

References

[1] R. Balzer, T. E. Cheatham, Jr., and C. Green, "Software Technology in the 1990's: Using a New Paradigm," Computer, pp. 39-45, November 1983.

[2] Judith E Grass, Yih-Farn Chen, "The C-- Information Abstractor", USENIX on C-- conference 1990.

[3] Emily Bark, Joseph Devlin, "Hypertext/Hypermedia Handbook", McGraw-Hill, 1991.

[4] F.Zdybel, N.R.Greenfeld, "An Information Presentation System", Proc.of IJCAI'81, pp. 978-984, August 1981.

[5] Moises Lejter, Scott Meyers, and Steven Po Reiss "Support for Maintaining Object Oriented programs,"IEEE Transaction on SE, Vol. 18, No. 12, pp. 1045-1052, Dec., 1992.

[6] M.F Kleyn and P. C. Gingrich, "Graph Trace Understanding Object-Oriented System Using Concurrently Animated View," OOPSLA, pp. 191-205, 1988.

[7] Mark A. Linton, John M. Vlissides, and Paul R.Calder "Composing User Interface with Interviews," Computer, Vol. 22, No.2, pp. 8-22. Feb., 1989.

[8] N. wilde and R. Huitt, "Maintenance Support for Object-Oriented Programs,", IEEE Translations on SE, Vol. 18, No. 12, pp. 1038-1044, Dec., 1992.

[9] Wyatt, B.B., Kavi,K., and Hufnagel,S. "Parallelism in Object-Oriented Lanuage: A Sur-vay," IEEE Software, pp. 56-66, November 1992.

[10] Haeng. K. Kim Ye. K. Son, "A Study on the Tool for Understanding the Constituent Information of Object-oriented System", Proceedings of the KISS conference, Vol. 20, No.1, pp. 525-528, 1993.

김 행 곤

1985년 중앙대학교 전자계산학과 졸업(학사)
1987년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1978년~1979년 미 항공우주국 객원연구원
1987년~1990년 한국전기통신공사 전임연구원
1988년~1989년 AT&T 객원연구원
1990년~현재 효성여대 전자계산학과 조교수
관심분야 : 객체지향시스템 설계, 사용자 인터페이스, 소프트웨어 재공학, 유지보수 자동화 툴, CASE

황 선 명

1982년 중앙대학교 전자계산학과 졸업(학사)
1984년 중앙대 대학원 전자계산학과 졸업(석사)
1987년 중앙대 대학원 전자계산학과 졸업(이학 박사)
1988년 독일 Bonn 대학 Informatik Ⅲ post doctor
현재~대전대학교 전자계산학과 조교수
관심분야 : 소프트웨어 품질보증 및 평가. 소프트웨어 테스팅