□특 집□

객체 지향 언어 C++를 위한 Information Viewer

건국대학교 전자계산학과 김문회* · 한재수

● 목 차 ●

Ⅰ. 서 론

하드웨어 기술의 급속한 발전과 가격 하락으로 컴퓨터의 응용 범위가 확산되어 소프트웨어의 수요가 폭발적으로 증가하고 소프트웨어의 복잡도와 크기도 기하급수적으로 늘어나고 있다. 이에따라 소프트웨어의 효과적인 개발이 요구되고 있으며 이를 위한 여러가지 방법론과 개발환경이 연구되고 있다. 1965년 E. J. Dijkstra의 구조적 프로그래밍을 시작으로 1960년대 후반부터 1970년대 초반에는 구조적 설계(structured design)[4], 정보은닉(Information hiding)[5], 모듈화(modularity) 등의 설계기법과 개념 등이 주로 연구되었고, 1970년대 후반에는 PSL/PSA(Problem Statement Language/Problem Statement Analyzer)[6], RSL/REVS(Requirement Statement Language/Requirement Engineering and Validation System)[7], SADT(Structured Analysis and Design Technique)[8], SSA(Structured System Analysis)[9, 10] 등의 요구분석기법과 지원도구들이 개발되었다. 1980년대 들어 Syntax-directed Editor, Knowledge-based Software Assistant (KBSA), Programmer's Apprentice (PA) 등의 지식 베이스 AI기법을 응용한 개발 방법론과 지원도구에 대한 연구가 시작되었다. 그러나, 여전히 소프트웨어 생산성이 수요를 충족시키기에는 요원하며, 이에 대한 타개책으로 1980년대 중반부터 객체지향 소프트웨어 개발 방법론 (OOP/OOD/OOA), 소프트웨어 재사용(reuse) 기법, CASE 등의 새로운 분야에 대한 연구가 현재 활발하게 진행되고 있다.

소프트웨어 생명주기 측면에서 볼 때, 가장 문제가 되고 있는 부분은 요구분석 단계와 유지 보수 단계이다. 이중에서도 유지 보수 단계에 드는 비용이 응용분야에 따라 다르지만, 일반적으로 전체 소프트웨어 비용의 60% 이상을 차지하고 있다. 일반적으로 유지 보수 단계에서 가장 문제가 되는 주요 원인은 주어진

*종신회원

소프트웨어에 내재하고 있는 복잡도에 있다. 이 복잡도에 영향을 주는 요소로는 소프트웨어 객체(예, 화일, 선언된 함수, 변수, 형선언문 등)들의 수, 객체와 객체간의 관계(예, 어떠한 함수가 어떤 변수를 사용하는가 등)의 양 등을 들 수 있다. 만일, 유지 보수자가 개발자가 아닐 때 주어진 소프트웨어를 유지 보수하는데 있어 이 내재된 복잡도를 극복하여 이해하는 것이 필수적이다.

본고에서는 소프트웨어 유지 보수에 도움을 줄 수 있는 도구인 Information Viewer 시스템의 전반적인 구성을 제시하고 현재 객체지향언어 중 보편적으로 사용되고 있는 C++를 대상으로 설계한 Information Viewer를 소개한다.

Information Viewer란 소프트웨어 개발 중 작성된 모든 문서(요구분석 명세서, 시스템 설계서, 소스코드 등)로부터 소프트웨어 유지 보수에 필요한 정보를 추출하여 소프트웨어 베이스(software base)에 저장하고, 유지 보수자가 필요한 정보를 필요한 시기에 용이하게 찾아볼 수 있도록 하여 주는 시스템이다. 이 시스템은 유지 보수자가 개발된 소프트웨어를 이해하는데 도움을 줄 수 있을 뿐만 아니라 소프트웨어 베이스의 활용을 통한 소프트웨어 재사용을 지원하고, 객체와 객체간의 관계를 분석함으로써 소프트웨어 시스템에서 사용되지 않는 부분(dead code)을 찾아내고, 소프트웨어 객체사이의 결합도(coupling)를 측정하는데 활용될 수 있다.

본고의 II장에서는 현재까지 진행되어 온 이 분야의 관련 연구를 소개하고, III장에서는 Information Viewer의 구성 개요를 기술하고 C++ 구문으로 재사용 지원, dead code 색출, 결합도 측정에 대한 예를 보인다. IV장에서는 현재 건국 대학교에서 개발한 C++를 위한 Information Viewer 프로토타입의 설계 및 구현을 기술하여 그 유용성을 보여주고 마지막으로 V장에서는 결론 및 앞으로의 연구과제를 기술한다.

II. 관련연구

여러 Information Viewer 시스템들이 문헌상에서 제안되고 보고되었다. 이 장에서는 그 중 실제로 구현된 시스템들에 대해 간단히 기술하고 이 구현된 시스템들을 검토하여 얻은 Information Viewer 시스템이 갖추어야 할 특성을 간단히 요약한다.

2.1 PIE browser

PIE browser[12]는 Smalltalk-80 browser[11]의 확장으로 사용자로 하여금 클래스 상속 계층을 검토할 수 있도록 하며, 시스템을 이해하는데 도움을 주기 위하여 텍스트 자료를 클래스에 연관시켜 줄 수 있는 기능이 있다. 또한 주어진 메시지를 어떤 클래스에서 받고 보내는지를 보여주는 기능도 있다. 이러한 종류의 browser들은 클래스의 모임이 작을 경우에 주로 시험되었고, 클래스의 모임이 커질 경우에는 그 유용성이 의문시 된다.

2.2 CATALOG

CATALOG[13]는 정보 검색 시스템으로 소프트웨어 재사용을 지원하는데 사용된다. 이 시스템은 관계 데이타베이스와 같이 정형화된 레코드나 텍스트와 같은 비정형화된 레코드를 함께 관리할 수 있으며, 사용자가 직접 추가하거나 수정 등을 할 수 있도록 한다. 레코드는 소프트웨어 모듈의 이름, 작성자, 간략한 기능설명 등을 저장한 주 레코드와 모듈코드 전체를 저장한 부 레코드로 구분된다. 찾고자 하는 소프트웨어 모듈의 기능이 질의를 통해 주어지면 data dictionary를 이용하여 질의에 포함된 단어와 유사한 단어까지 고려하여 데이타베이스를 검색하고 그 결과를 사용자에게 보여준다. 소프트웨어의 재사용을 더욱 효과적으로 하기 위해서는 모듈과 함수의 이름, 작성자, 기능설명 등을 구조적 comment기법과 같은 방법을 이용하여 정형화하는 것이 필요하다.

2.3 MasterScope

MasterScope는 Interlisp[14] 환경하에서 사용자 프로그램을 분석하고 상호참조(cross-reference)하는데 사용된다. MasterScope는 함수 정의를 분석하고 함수들의 호출에 대한 트리구조를 인쇄하는 PrintStructure라는 간단한 프로그램에서 시작되었다. 그후에 PrintStructure에 의해서 생성된 많은 자료로부터 특정한 정보를 발췌하는데 어려움이 없도록 하기 위하여 프로그램 분석과 질의를 구분하였다. MasterScope는 Interlisp에서 화일과 편집 패키지를 가지고 있으며 이것은 자동적으로 프로그램의 변경된 부분을 재분석함으로써 데이타베이스가 유지 보수되도록 한다. 사용자는 영어와 명령언어를 사용하여 MasterScope와 대화한다. Common Lisp Framework (CLF) Programming 환경은 MasterScope와 유사한 점적인 분석

도구가 있다.

2.4 FAST

FAST(Fortran Analysis System)[15]는 Fortran 프로그램을 위한 분석능력을 제공한다. MasterScope와는 달리 FAST는 계층 데이타베이스를 이용한 프로그램 정보 관리 시스템이다. FAST는 이 방식으로 시스템에서 함수의 중복을 피하도록 도와준다. FAST의 개념적 모델은 모듈의 속성, statement, 이름과 언어 객체들의 관계로 구성되어 있다. 또한, FAST는 질의와 분석기능을 가지고 있는 별도의 명령언어를 사용한다.

2.5 OMEGA

프로그램의 정보를 데이타베이스에 저장하려는 생각은 실험 시스템인 OMEGA[16]에서 구현되었다. OMEGA는 자료관리를 위해 INGRES를 사용하여 프로그램 데이타베이스에 모든 관계 연산자를 적용할 수 있다. OMEGA의 목적중의 하나는 프로그램 데이타베이스로부터 소프트웨어 객체의 재구성을 하는 것이다. 따라서 변수, experession, statement, 그리고 이들 사이의 관계에 대한 상세한 정보가 데이타베이스에 저장되었다. 58개의 관계가 데이타베이스 스키마에서 사용되었다. 이러한 정보는 처리하는데 많은 시간이 걸리고 작은 소프트웨어에도 커다란 데이타베이스를 운영하게 된다. OMEGA의 프로토타입 구현을 보면 프로시저 부분을 검색하는데 많은 시간이 소요된다.

2.6 Cscope

Cscope[17]는 사용자로 하여금 C 심볼의 사용도를 검사하고 함수정의와 매크로를 찾거나 수정하게 하는 화면을 이용한 C program browser이다. Cscope는 모든 소스텍스트를 압축된 형태로 저장하고 있는 상호 참조 화일을 만든다. 각 라인에는 심볼에 대한 참조 정보가 부가적으로 첨가되고 상호 참조 화일의 크기는 소스화일의 크기와 비슷하다.

2.7 CIAS

CIAS(C information abstractor system)[18]는 큰 C 프로그램으로부터 객체 및 객체간의 관계에 관한 정보를 추출하여 관계 데이타베이스에 저장한다. 저장된 정보는 사용자의 요청에 따라 정보 viewer에 의해 테이블 형식으로 보여진다.

2.8 검토사항

앞에 기술한 시스템을 검토함으로써 얻어진 Information Viewer 시스템이 갖추어야 할 바람직한 특성은 다음과 같다.

- 소프트웨어 모듈의 분류화 : 관리하여야 할 자료와 정보를 보관하는 데이타베이스의 크기가 커질 경우 검색시간이 많이 소요되고, 자료의 관리에도 비효율적임을 PIE browser나 OMEGA의 경우에서 볼 수 있었다. 이러한 문제점을 극복하기 위해서는 소프트웨어 모듈을 기능별, 응용분야 등에 따라 분류하여 보관하는 것이 필요하다. 하지만 이 분야에 대한 연구는 미미한 실정이다.
- 정보 추출과 검색과정의 분리 : 정보를 추출하여 보관하는 과정과 보관된 정보를 질의를 통해 검색하는 과정을 별도로 분리하는 것이 검색과정에 큰 융통성을 준다. Interlisp에서 PrintStructure를 MasterScope로 확장한 것도 이러한 원칙을 적용하기 위한 노력의 일환이다. OMEGA, FAST, 그리고 CLF도 이러한 원칙에 의해 개발되었다.
- 소프트웨어 정보 베이스(Software information base: SIB)와 문서 데이타 베이스(document database)들의 분리 : 소프트웨어 개발과정 중에 작성된 소스 프로그램을 포함한 모든 문서로 이들로부터 추출된 유지 보수에 필요한 소프트웨어 정보를 하나의 데이타베이스에 보관하는 것은 비효율적이다. 요구분석 명세 데이타베이스, 설계 명세 데이타베이스, 소스 프로그램 데이타베이스 등을 따로 분리하고 이들로부터 추출된 소프트웨어 객체 및 객체간의 관계에 대한 정보는 따로 소프트웨어 정보베이스에 보관하는 것이 자료관리에 효율적이다.
- 명확한 개념적 모델 설정 : 개념적 모델은 어떠한 정보가 추출되고 보관되어야 하는지를 정의한다. 이 개념적 모델은 문서의 구조, 관리방식, 그리고 사용된 프로그래밍 언어에 따라 달라지지만, 대부분의 경우 entity-relationship 모델이 적용될 수 있다. III 장에서 C++ 프로그램에 대한 개념적 모델을 entity-relationship 모델로 표현한 예를 보여준다.

이상의 검토사항들은 다음 장에 기술한 Informa-

tion Viewer 시스템의 설계에 대한 논리적 근거이다.

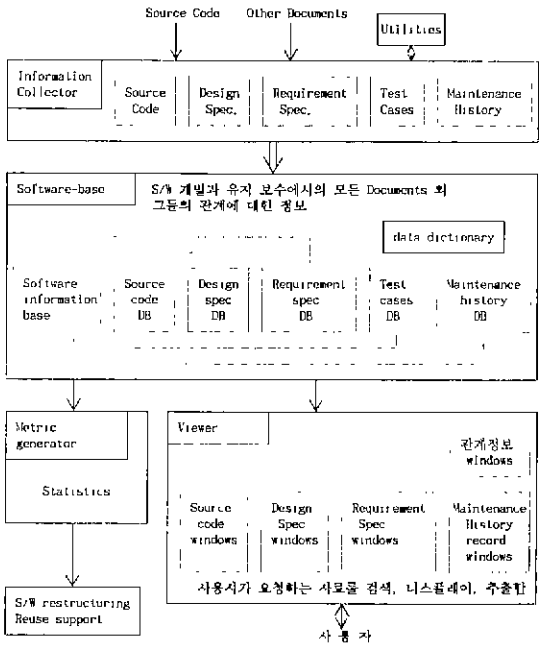
III. Information Viewer 시스템

본고에서 제시하는 Information Viewer 시스템은 (그림 1)에서 보듯이 4 구성요소로 이루어져 있다: Information collector, Software-base, Metric generator 그리고 Viewer. Information collector는 소프트웨어 생명주기 단계에서 생성되는 모든 문서로부터 정의되는 객체와 그 객체들의 관계를 추출하여 Software-base 안에 있는 Software information base(SIB)에 저장한다. 여기서 객체란 소스 프로그램내에 정의된 변수, 함수뿐만 아니라 그에 관련된 요구분석 명세, 설계명세, test case 등을 포함한다. Software-base에는 Information collector에서 추출한 정보 뿐만 아니라 생성된 모든 문서가 보관된다. Metric generator는 Software-base를 검사하여 소프트웨어 객체들과 객체간의 관계를 근거로 보관된 소프트웨어 시스템들의 복잡도를 측정하며 Viewer는 사용자의 요청에 따라 Software-base로부터 필요한 정보를 검색하여 보여주거나 추출한다. 이러한 시스템은 유지 보수하는 과정에서 필요한 정보를 신속하게 제공하고 수정하고자 하는 소프트웨어 객체와 그와 관련된 다른 객체를 일관성있게 처리할 수 있도록 함으로써 유지 보수 단계에서의 오류를 방지하는데 도움을 준다. 또한 새로운 소프트웨어를 개발할 때 소프트웨어 베이스를 검색하여 필요한 부분을 재사용할 수 있게 함으로써 생산성과 신뢰도를 높이는데 도움을 줄 수 있다.

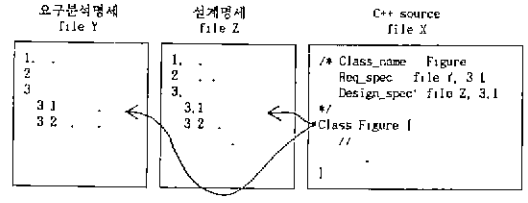
다음 절에 Information viewer 시스템을 구성하는 4 요소들의 기능을 설명한다.

3.1 Information collector

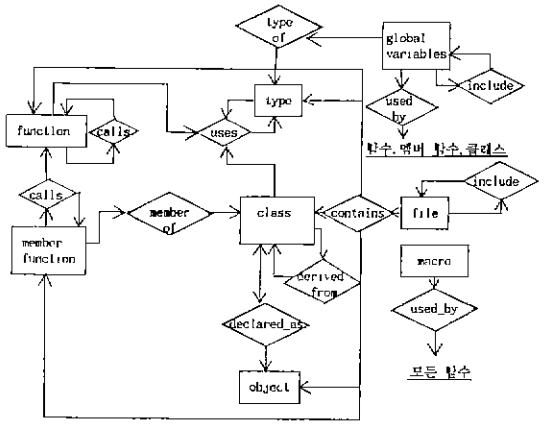
요구분석 명세서, 설계 명세서, 소스 프로그램, test cases, 유지 보수 기록 등 소프트웨어 생명주기 단계에서 생성되는 모든 문서에 포함되어 있는 정보를 수집한다. 이를 용이하게 하기 위해서는 형식언어(formal language)의 사용이 바람직하나 형식언어의 사용이 어려운 경우에도 각 문서는 구조적으로 구성되어 있어야 한다. 예를들면, (그림 2)는 비형식언어로 쓰여진 문서의 내용이 장·절 등의 구조로 표시되어 객체간의 관계가 구조적 comment를 통하여 표시될 수 있음을 보여주고 있다. (그림 2)에서는 Class_



(그림 1) Information viewer 시스템의 구성도



(그림 2) 객체간의 관계 자료 표시



(그림 3) C++ 프로그램의 개념적 모델

name, Req_spec 등은 Information collector가 인식할 수 있는 키워드라고 가정하고 class Figure와 관련된 요구분석 명세와 설계 명세의 위치를 이 키워드를 통해 표시하고 있다.

Information collector에서 수집하는 정보는 대상 프로그래밍 언어에 따라 달라진다. 대상 프로그래밍 언어가 C++인 경우에 수집되는 정보는 (그림 3)의 개념적 모델에서 개략적으로 보여주고 있다.

수집되는 정보는 3가지 종류로 구분할 수 있다.

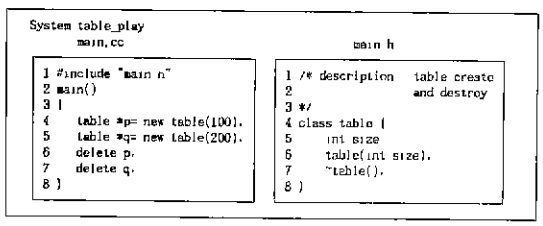
• 객체 : (그림 2)와 (그림 3)에서 볼 수 있듯이 모든 문서에 포함된 객체의 이름을 수집하고 객체의 종류를 구분한다. (그림 4)에서 보면 다음과 같은 정보가 수집된다.

system: table_play
file: main.cc, main.h
class: table

Information collector로 main 함수안에 선언된 객체 p와 q에 대한 정보도 필요에 따라 수집될 수 있도록 하는 것이 바람직하나 기본적으로 전역 객체(global object)들에 대한 정보만을 수집한다. 지역 객체(local object)에 대한 정보를 수집하게 되면 수집되는 정보의 양이 너무 커져 비효율적이 된다. 보다 효율적인 Information collector는 수집하는 정보의 범위를 사용자로 하여금 정의하도록 하여 사용자에 의해 요청된 정보만을 수집하는 것인데 어떻게 사용자가 Information collector에 수집할 정보의 범위를 규정하고 Information collector가 요청된 정보만을 수집하게 하는가에 대한 연구가 선행되어야 할 것이다.

• 객체의 속성 : 각 객체는 관련된 속성들을 가지고 있다. (그림 4)의 table에 대한 속성은 다음과 같다.

file: main.h
object type: class
name: table



(그림 4) 간단한 C++ 프로그램

begin line: 4
end line: 8
description: table, create, destroy

이 예에서 보듯이 객체에 대한 설명이 구조적 comment형식(description)으로 주어지면 중요 단어(명사, 동사 등)만을 추출하여 보관한다.

• 객체간의 관계 : (그림 2)와 (그림 3)에 나타나 있듯이 각 객체간의 상관관계가 수집된다. (그림 4)의 예에서 보면 다음과 같은 관계가 수집되며 이 객체간의 역 관계도 성립된다.

객체 1	관 계	객체 2	코멘트
main.cc	include	main.h	file to file
main.cc	contains	main	file to function
main.h	contains	table	file to class

3.2 소프트웨어 베이스

소프트웨어 베이스는 각종 문서와 Information collector에서 수집된 다양한 정보를 체계적으로 보관하고 관리하여야 한다.

소프트웨어 베이스는 기본적으로 세 부분으로 나눌 수 있다.

• SIB: Information collector에서 수집한 정형화된 정보를 보관하는 장소로써 각 문서 DB상에 보관되어 있는 객체들을 연결하여 주는 중추적인 역할을 한다.

• Data dictionary: Viewer에 사용자가 자료를 요청하였을 시 요청에 포함된 단어와 유사한 단어도 검색될 수 있도록 각 단어의 동의어를 규정하고 있다.

• 각 문서 DB: 요구분석 명세, 설계 명세, 소스 코드 등의 비정형화된 자료를 구조적으로 보관한다.

3.3 Metric generator

Metric generator는 소프트웨어 시스템에 포함되어 있는 객체의 수, 객체간의 관계수, 객체의 사용 빈도수 등을 소프트웨어 베이스로부터 분석하여 사용자에게 보여줌으로써 소프트웨어 시스템에 내재하고 있는 복잡도를 측정하는데 도움을 준다. 또한 dead code의 검출, 시스템 재구성(restructuring)에도 도움을 준다. 예를들어, C++ 프로그램의 경우 어떠한 class가 다른 class를 derive하는데 사용되지 않았고 그 class의 사용 빈도수가 0이면 이 class에 대한 정의는 불필요함을 알 수 있다. 또한, file A에 정의되어진 class

X가 file A에서는 사용되지 않고, file B에서만 사용되었다면 file A와 file B간의 관계수가 높아질 것이며 class X의 정의는 file B로 이동되야 함을 알 수 있을 것이다.

3.4 Viewer

Viewer는 사용자로부터 요청을 받아 소프트웨어 베이스 관리 시스템이 인식할 수 있는 언어로 변환하여 소프트웨어 베이스 관리 시스템에 보내고, 그 결과를 사용자 인터페이스를 통하여 사용자에게 보여준다.

사용자 인터페이스는 window를 이용하여 정보를 도식화하여 쉽게 볼 수 있도록 하는 방법이 많이 개발되고 있다. 이러한 window는 사용자가 정의된 명령어를 이용한 질의에 의해서 정보를 검색하는 방법과 마우스의 간단한 조작으로 화면에 사용자가 필요한 자료를 직접 검색하여 내용을 도식적으로 볼 수 있도록 하는 방법 등 여러가지 방법으로 사용자가 편리하게 이용할 수 있도록 많은 연구가 진행되고 있다.

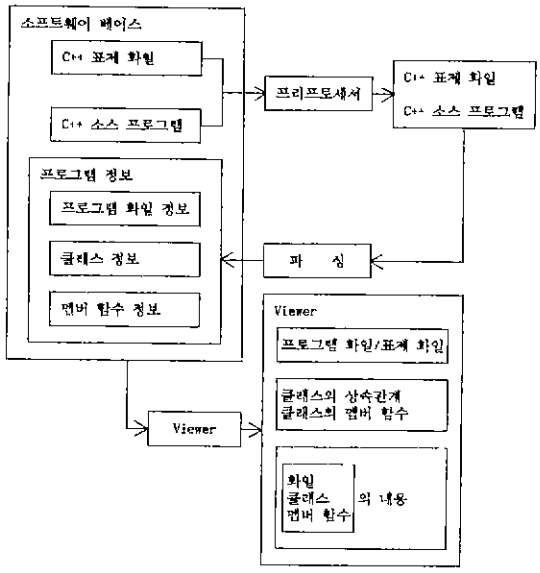
IV. C++를 위한 information viewer 프로토타입

앞 장에서 제시한 Information viewer 시스템의 유용성을 검증하기 위하여 C++ 소스 프로그램을 대상으로한 프로토타입이 건국대학교에서 개발되고 있다. C++는 현재 가장 보편적으로 사용되고 있는 객체지향 언어로써 다음과 같은 객체지향 개념을 지원하고 있다.

- 객체(object)
- 클래스(class)
- 상속성(inheritance)
- 다형(polymorphism)
- 동적 바인딩(dynamic binding)

이 개념들의 정의는 여러 참고 문헌에서 찾아볼 수 있다[23, 28].

다형과 동적 바인딩에 관한 정보는 소스 프로그램을 분석하여 얻을 수 없고 프로그램을 실행하여야 얻어질 수 있으므로 프로토타입에서는 고려되지 않았고 C++의 새로운 기능들인 template라 불리우는 파라메터화된 type 기능과 사용자 정의 exception handling 기능에 대한 정보도 프로토타입에서는 고려되지 않았다. 프로토타입의 신속한 개발을 위하여 클래스와



(그림 5) C++ Information viewer 프로토타입의 구조도

상속성에 관련된 정보만이 고려되었다.

객체지향 개념의 상속성은 재사용성과 확장성을 효율적으로 구현하기 위한 것이다. 그러나 상속성이 잘못 남용되거나 시스템내에 상속계층(inheritance hierarchy)의 레벨이 너무 깊어지면 프로그램의 복잡도를 증가시킬 수 있으므로 프로토타입을 통해 상속성으로 인한 어려움을 극복하는데 얼마나 도움을 주는가를 실험하는데 촛점을 맞추었다. (그림 5)는 이 프로토타입의 구조를 보여주고 있으며 구현환경, 정보수집절차, viewer 및 실험결과를 다음 절에서 기술한다.

4.1 프로토타입 구현 환경

프로토타입은 SUN4/280을 서버로 하는 SUN3/50 워크 스테이션에서 개발되었고 세 가지 언어(FLEX, BISON, C++)로 쓰여져 있으며 총 3,588줄로 구성되어 있다. 프로토타입에서는 수집된 정보를 데이타베이스 시스템에 저장하는 대신 SUN 화일에 저장하였다. 사용자 인터페이스는 X window가 이용되었으며 Motif를 이용하여 window를 구성하였다.

4.2 정보수집

C++ 프로그램은 일반적으로 여러 화일로 나뉘어져

있다. 따라서 여러 화일로 나뉘어진 모든 화일을 파싱함으로써 클래스와 그의 함수에 대한 정보를 수집하게 된다. 정보를 수집하는 절차는 다음과 같다.

가. C++ 프리 프로세서(pre-processor)를 이용하여 표제 화일(header file)의 내용을 프로그램에 포함시킨다. 따라서 표제 화일에 대한 별도의 처리가 불필요하다.

나. C++ 프리 프로세서에 의해서 생성된 화일을 모두 파싱한다. C++ 프로그램에 대한 파서는 FLEX와 BISON으로 구현되었다.

다. 화일을 파싱하는 동안 필요한 정보를 수집하여 저장한다.

(1) C++ 프리 프로세서에 의해서 생성된 화일은 일정한 규칙에 의해서 프로그램 화일과 표제 화일을 구분할 수 있다. 이 규칙을 이용하여 프로그램 화일과 표제 화일을 수집하여 이의 관계를 레벨로 표시하여 화일간의 관계(dependency)를 나타낸다. 프로그램에서 정보는 프리 프로세서에 의해서 생성되기 전의 화일의 프로그램 라인을 이용하여 정보의 위치를 지정하므로 화일의 정보를 수집할 때 화일의 이름과 라인의 수를 다시 지정한다.

(2) 클래스에 대한 정보는 클래스의 이름과 화일에서의 라인의 위치를 수집하여 클래스의 시작 라인과 종료 라인을 저장한다. 다른 프로그램에서 같은 표제화일을 사용할 수 있으므로 클래스가 먼저 정의되어 있는지를 우선 찾아보고 정의되어 있지 않으면 저장을 한다.

(3) 클래스의 상속관계는 다중 상속관계를 포함하여 하나의 클래스에 대한 상위 클래스의 정보를 수집하고 수집된 자료를 모두 연결하여 모든 클래스의 관계를 나타낼 수 있도록 한다.

(4) 클래스의 멤버 함수는 멤버 함수의 이름과 함께 함수의 인수들에 대한 자료형을 포함함으로써 같은 이름의 중복 함수(overloaded function)를 구분한다.

(5) 클래스의 멤버 함수는 클래스에만 있는 경우와 클래스에는 멤버 함수의 이름과 인수만 있고 클래스의 외부에 멤버 함수가 있는 경우가 있다. 이것은 클래스와 멤버 함수관계의 정보를 하나의 화일에 저장하고, 멤버 함수의 위치에 대한 정보는 다른 화일에 저장하여, 두 가지 경우를 같은 방법으로 처리한다.

4.3 소프트웨어 베이스의 구성

프로토타입에서는 소프트웨어 베이스로 SUN 화일이 사용되었으며 소스 프로그램에서 수집된 정보는 각각의 자료 형식에 따라 화일에 다음과 같은 형태로 저장된다.

가. 소스 프로그램과 표제 화일의 관계

형식 : File Level-File Name

예) 1-man.cc
2-screen.h
2-myshape.h
3-shape.h

나. 클래스가 속한 화일명과 위치

형식 : File/Class Name/Begin/End

예) shape.h/rectangle/40/56

다. 클래스의 상속관계

- 형식 1 : Class Name/Access Spec./Parent Class Name

  예) rectangle/public/shape
- 형식 2 : Class Name/Child Class Name

  예) shape/rectangle

라. 클래스내의 함수관계

- 형식 1 : Class Name/Access Spec./Function Name

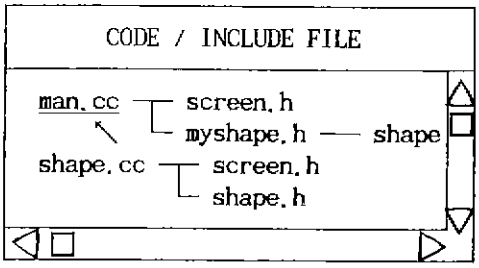
  예) rectangle/public/rectangle(point, point)
  rectangle/public/draw( )
- 형식 2 : File/Class Name/Function Name/Begin/End

  예) shape.cc/rect./rectangle::rectangle(point, point)/4/32
  shape.cc/rectangle/rectangle::draw( )/34/42

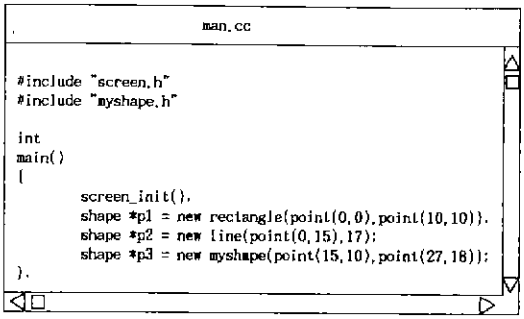
4.4 Viewer

X-window에서 Motif를 이용하여 소프트웨어 베이스에 있는 자료를 검색하여 화면에 보여주는 Viewer를 구현하였다. 초기 window는 화일 계층 window와 클래스 계층 window로 나누어져 있으며 아래에 설명되는 다른 window는 선택에 따라 pop-up 된다.

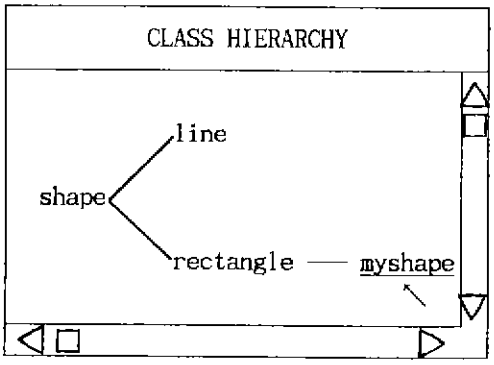
가. 소프트웨어 베이스에 있는 화일에 대한 정보에 의해 window에 프로그램 화일과 표제 화일의 관계를



(그림 6) C++ 화일 계층 window



(그림 7) C++ 화일 계층 내용 window

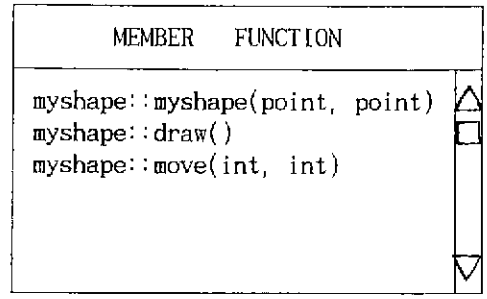


(그림 8) C++ 클래스 계층 window

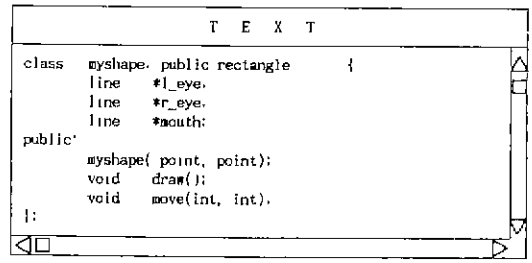
보여준다. (그림 6)과 같이 이 window에서 화일명을 선택하면 그 화일의 내용을 (그림 7)과 같이 보여준다.

나. 소프트웨어 베이스에 있는 클래스의 상속관계를 보여준다. 상위 클래스와 하위 클래스는 선으로 연결하여 관계를 표현하며 왼쪽이 상위 클래스이다(그림 8).

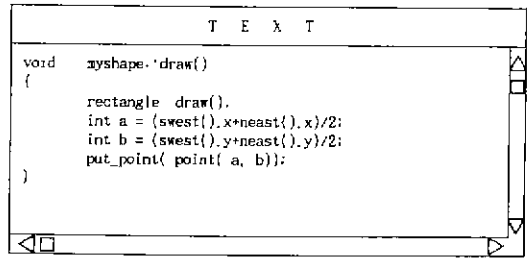
다. 클래스를 선택하면 이 클래스의 멤버 함수와 클래스의 내용이 각각 화면에 나타난다. 클래스의 내용은 저장되어 있는 화일을 찾아서 시작과 종료를 나타내는 수를 이용하여 내용을 읽고 이것을 화면에 보여준다. (그림 8)의 클래스 계층에서 클래스 이름을



(그림 9) C++ 멤버 함수 window



(그림 10) C++ 클래스 내용 window



(그림 11) C++ 멤버 함수 내용 window

선택하면 멤버 함수 window에 멤버 함수 이름이 모두 리스트되고(그림 9), window 하단에는 클래스의 내용이 모두 나타난다(그림 10).

라. 클래스에 속한 멤버 함수를 선택하면(그림 9) 멤버 함수 내용이 모두 나타난다(그림 11).

이와 같이 클래스 상속관계를 도식적으로 나타내고 상세한 내용은 마우스를 이용하여 보여줌으로써 프로그램의 복잡한 클래스간의 관계를 쉽게 알 수 있으며, 이 정보를 이용하여 프로그램의 추가, 수정 등의 유지 보수를 쉽게 할 수 있다.

4.5 실험 평가

두 개의 C++ 소스 프로그램을 대상으로하여 프로

〈표 1〉 프로토타입 실험 결과

	프로그램 1	프로그램 2
Line 수	17,383	12,569
클래스 수	37	31
멤버 함수 수	474	404
화일 수	3	3
클래스 계층 정보 수	24	22
수집된 관계 정보 수	1,394	1,197
처리시간 (초)	37	26

토타입의 성능을 실험하였고, (표-1)은 그 결과를 보여주고 있다. (표-1)에서 프로그램 1은 M/M/1 Queue를 simulation한 것이고 프로그램 2는 dining philosophers에 대한 것이다. 이 프로그램들은 [31]에 자세히 기술되어 있다.

(표-1)에서 보듯이 두 프로그램으로부터 수집된 클래스 계층정보를 포함한 관계정보의 수는 각각 1394, 1197이며 이만한 정보를 수집하는데 걸린 시간은 각각 37초, 26초이다. 짧은 시간에 큰 프로그램으로부터 정보를 수집할 수 있었던 원인은 수집한 정보의 종류가 제한되어 있었기 때문이며 모든 필요한 정보를 수집할 경우의 성능은 앞으로 프로토타입을 확장하여 실험할 계획이다.

위와 같이 생성된 정보를 가지고 두 프로그램을 이해하는데 사용해 보았는데 많은 도움이 있었다는 것을 정량적으로 표현할 수는 없지만 이 프로토타입을 사용해 본 실험자(4명)의 공통적인 의견이었으며 프로토타입이 확장되어 더욱 많은 정보를 수집하게 되면 그 유용성이 현격히 증대할 것으로 사려된다.

V. 맺음말

본고에서는 소프트웨어 유지 보수에 도움을 주고 소프트웨어 재사용을 용이하게 하여 주는 도구인 information viewer 시스템을 제시하고 객체지향언어인 C++로 쓰여진 프로그램을 대상으로 개발한 프로토타입의 설계, 구현 및 실험결과를 기술하였다. 소프트웨어 생산성을 획기적으로 높이기 위해서는 객체지향 소프트웨어 개발 방법론 등의 새로운 개발 방법론과 아울러 이러한 지원도구의 개발이 필수적이라 하겠다. 끝으로 C++ information viewer 프로토타입의 확장에 대한 현재 추진방향은 다음과 같다.

• 소프트웨어 베이스로 POSTGRES나 객체지향 DBMS의 적용여부 검토

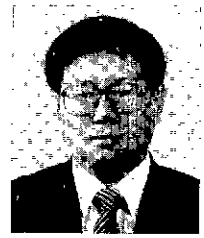
• C++ Information viewer 프로토타입의 정보수집기능 확장과 그에 따른 Viewer의 개선

• Metric을 이용한 시스템 재구성에 관한 연구 및 프로토타입 개발

참 고 문 헌

1. P. Coad and E. Yourdon, "Object-Oriented Design," Prentice-Hall, 1991.
2. B. J. Cox and A. J. Novobilski, "Object-Oriented Programming: An Evolutionary Approach," Addison-Wesley, 1991.
3. S. A. Dart, R. J. Ellison, P. H. Feiler and A. N. Harbermann, "Software Development Environments," IEEE Computer, Nov. 1987, pp. 18~28.
4. W. Stevens, G. Myers, and L. Constantine, "Structured Design," IBM Systems Journal, Vol. 13, No. 2, 1974.
5. D. Parnas, "Information Distribution Aspects of Design Methodology, IFIP Congress Proceedings, 1971.
6. D. Teichrow and E. Hershey, "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," Trans. Software Eng., Jan. 1977.
7. M. Alford. "A Requirements Engineering Methodology for Real-Time Processing Requirements," Trans. Software Eng. Jan. 1997.
8. D. Ross, "Structure Analysis (SA): A Language for Communicating Ideas," Trans. Software Eng., Jan. 1977.
9. C. Gane and T. Sarson, "Structured System Analysis: Tools and Techniques," Prentice-Hall, 1979.
10. T. DeMarco. "Structured Analysis and System Specification," Yourdon Press, 1978.
11. A. Goldberg, "Smalltalk-80: The Interactive Programming Environment," Addison-Wesley, 1984.
12. I. P. Goldstein and D. G. Bobrow, "A layerd approach to software design," Rep. CSL-80-5, Xerox, 1980.
13. W. B. Frakes and B. A. Nejmeh, "Software Reuse Through Information Retrieval," Proc. 20

th Annu. Hawaii Int. Conf. on Sys. Sciences, 1987, pp. 530~535.
14. W. Titleman and L. Masinter, "The Interlisp programming environment." Computer, Vol. 4, No. 4, Apr. 1980, pp. 25~34.
15. C. Browne and D. B. Johnson, "FAST: A second generation programming analysis system," Proc. 2nd Int'l Conf. Software Eng., 1977, pp. 142~148.
16. M. A. Linton, "Implementing relational views of programs," Proc. ACM SIGSOFT/SIGPLAN Software Eng. Symp. Practical Software Development Environment, May 1984.
17. J. L. Steffen, "Interactive examination of a C program with Cscope," Proc. USENIX Assoc. Winter Conf., Jan. 1985, pp. 170~175.
18. Y. F. Chen, M. Y. Nishimoto, and C. V. Ramamoorthy, "The C Information Abstraction System," IEEE Trans. Software Eng., Vol. 16, No. 3, Mar. 1990, pp. 325~334.
19. S. Gibbs. D. Tsichritzis, E. Casais, O. Nierstrasz, and X. Pintado "CLASS Management For Software Communities," Commun. ACM, Sept. 19 90, pp. 90~103.
20. R. Helm and Y. S. Maarek, "Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries," OOPSLA, 91, pp. 47~61.
21. B. Henderson-Sellers and J. M. Edwards, "The Object-Oriented Systems Life Cycle," Commun. ACM, Sept. 1990, pp.. 104~124.
22. D. Kafura and G. R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," IEEE Trans. Software Eng., Mar. 1987, pp. 335~343.
23. T. Korson and J. D. McGregor, "Understanding Object-Oriented: A Unifying Paradigm," Commun. ACM, Sept. 1990, pp. 41~53.
24. A. V. Lamsweerde, B. Delcourt, E. Delor, M. Schayes, and R. Champagne, "Generic Lifecycle Support in the ALMA Environment," IEEE Trans. Software Eng., Jun. 1988, pp. 720~741.
25. M. H. Penedo and W. E. Riddle, "Guest Editor's Introduction: Software Engineering Environment Architectures," IEEE Trans. Software Eng., Jun. 1988, pp. 689~696.
26. R. S. Pressman, "Software Engineering: a practitioner's approach," 3rd ed., McGraw-Hill, 1992.
27. A. T. Schreiner and H. G. Friedman Jr., "Introduction to Compiler Construction with UNIX," Prentice-Hall, 1985.
28. B. Stroustrup, "The C++ Programming Language," Addison Wesley, 1991.
29. R. J. Wirfs-Brock and R. E. Johnson, "Surveying current research in object-oriented design," Commun. ACM, Sept. 1990, pp. 104~124.
30. D. A. Young, "The X Window System: Applications and Programmung with Xt OSF/Motif Edition," Prentice-Hall, 1990.
31. D. Grunwald, "A Users Guide to AWESIME: An Object Oriented Parallel Programming and Simulation Systme," Tech. Report CU-CS-552-91, Dept. of Computer Science, University of Colorado at Boulder, Nov. 1991.



김 문 회

1975 ~1979 서울대학교 공과대학 전기공학과, B.S
1979 ~1981 서울대학교 공과대학 전기공학과, M.S
1983 ~1985 University of South Florida, M.S in Computer Science
1986 ~1991 University of California, Berkeley, Ph.D.
1986 ~1991 UCB ERL Post Graduate Researcher (연구원)
1991 ~현재 건국대학교 전자계산학과 조교수
관심 분야 : 소프트웨어 공학, Real-time distributed computing system, Fault-tolerant computing system, Network management

한 재 수

1973 ~1977 연세대학교 이과대학 수학과, B.S
1977 ~1982 삼성전기 정보시스템실
1983 ~1989 기아 자동차 전산부
1991 ~1993 건국대학교 공과대학 전자계산학과, M.S
관심 분야 : 소프트웨어공학, 객체지향방법론