論 文

大 韓 造 船 學 會 論 文 集
第 34 卷 第 4 號 1997年 11月
Journal of the Society of
Naval Architects of Korea
Vol. 34, No. 4, November 1997

데이터베이스와 응용프로그램 사이의 인터페이스 설계 및 구현

정인숙**, 이창섭*, 조충호**

Design and Implementation of Interface between the Database and the Application Program

by

In-Sook Jung**, Chang-Sup Lee* and Chung-Ho Cho**

요 약

DOS 환경하에서 개발된 기존의 대부분의 응용프로그램들은 ASCII 형식의 파일을 직접 읽고 씀으로써 외부 세계와 정보를 교환한다. 수많은 단위 모듈로 이루어지는 종합적인 패키지는 모듈 각자의 입출력뿐아니라 모듈 사이의 빈번한 자료교환을 수행하여야 하므로, 모듈의 수가 증가할수록 ASCII 파일을 통한 입출력은 비효율적이고 정확성도 결여된다. 본 논문에서는 응용프로그램이 필요로 하는 모든 정보를 데이터베이스에 저장하고, 패키지의 각 모듈이 데이터베이스와 직접 정보를 교환할 수 있는 인터페이스를 설계하고 이를 실제 구현하는 과정을 보이고 있다. 개발된 인터페이스를 기존의 프로펠러 설계-해석 시스템에 적용하여 데이터베이스와의 정보교류가 원활히 수행되고 있음을 보였다. 이상을 Unix 환경하의 워크스테이션이 아닌 Windows95 환경하의 개인용/휴대용 컴퓨터에서 구현함으로써 프로펠러의 설계-해석을 시간적 공간적 제약없이 수행할 수 있음을 보였다.

발 표 : 1997년도 대한조선학회 춘계연구발표회('97. 4. 26)
접수일자 : 1997년 4월 28일 재접수일자 : 1997년 8월 24일
* 정회원, 충남대학교 공과대학 선박해양공학과
** 학생회원, 충남대학교 공과대학 선박해양공학과

Abstract

Most of the existing applications developed undter the DOS environment exchange information with the exterior world through reading and writing files in ASCII format. A package program consisting of numerous modules has to exchange data not only between module program and the input/output files but also between module themselves, and hence information exchange via ASCII file is increasingly inefficient and less accurate as the number of modules increases. The present paper describes the design and the implementation of the interface between the module program and the database which stores all the information necessary for the execution of the application. The interface program is then applied to the existing Propeller Design and Analysis System to show that the interface functions smoothly. The development is performed on the personal computer operating under Windows 95 rather than the workstation under Unix operating system to prove that the design and analysis work can be carried out without limit in time and space.

1. 서 언

설계, 해석을 위한 종합적인 프로그램 패키지는 수 많은 계산 단계를 필요로 하며 이에 따라 각 단계를 계산할 수 있는 기본적인 모듈로 이루어지게 된다. 일반적으로 각 모듈의 입출력을 원활히 따라가는 것은 각 모듈의 개발자 또는 숙련된 사용자라도 쉬운 일이 아니며, 각 모듈의 사용법이 서로 복잡하게 연결되어있는 종합적인 패키지의 경우는 상호 정보의 교환을 동시에 파악하는 것이 매우 어려운 문제이다. 이러한 종합적인 패키지를 쉽고 효과적으로 활용하기 위하여는 사용자가 패키지의 흐름을 쉽고 정확하게 파악할 수 있어야하고, 계산의 결과를 간단하면서도 정확하게 판단하고 제시할 수 있어야하는 동시에, 프로그램 패키지를 구성하고 있는 각 모듈 사이의 정보의 교류를 쉽고 정확하게 관리할 수 있어야 한다.

복잡한 패키지의 흐름을 쉽게 파악하며 원하는 설계, 해석을 수행하기 위하여는 그래픽 사용자 인터페이스(graphic user interface, GUI)를 사용하는 것이 널리 알려져 있다. 조선분야에서의 응용은 선박설계생산전산화(CSDP) 사업에서 워크스테이션에서 작동하는 시스템을 개발한 것이 잘 알려져 있으며, 개인용 또는 휴대용 컴퓨터에서의 설계 및 해석에 적용된 예는 프로펠러 설계 및 해석 시스템(Propeller Design and Analysis System, ProDAS)[1]이 알려져 있다. 설계, 해석의 결과를 시각적인 효과를 극대화하여 쉽고 정확하게 파악하기 위하여는 이러한 GUI로 구동되는 패키지에 그래픽을 이용한 출력을 최대한 활용하는 것이 바람직하다. 이를 위하여 상용화된 그래픽 패키지를 동적연결방식(Dynamic Link Library, DLL)으로 GUI 제어시스템에 통합하면 원하는 결과를 마우스를 클릭하는 것만으로도 간단히 확인하는 것이 가능하다. 이러한 개념을 개인용 컴퓨터에 적용한 예는 이창섭등[2]에 발표된 바 있다.

각 모듈 자체의 계산을 위한 입력자료 뿐만아니라 각 모듈 사이의 정보 교류를 원활히 하기 위하여는 정보가 체계적으로 저장, 관리되어야 하며, 정보의 사용자인 각 프로그램 모듈로부터의 접근이 용이해야 한다. 본 논문은 요소 프로그램 모듈과 정보 데이터베이스 시스템 사이의 연결을 체계적으로 정립하는 과정을 기술하고 있다.

이를 위하여 우선 기존의 패키지들에 포함되어 있는 대부분의 계산 프로그램들이 데이터베이스를 활용하지 않고 있거나, 활용하더라도 데이터베이스와 요소 프로그램 사이에 ASCII 파일을 활용하고 있다는 사실을 지적하고자 한다. ASCII 파일의 활용은 전통적인 입출력 방식을 그대로 간직하고 있는 것으로, 패키지가 대형화 할수록 필요없는 파일을 양산하고 데이터베이스를 중복 관리, 저장함으로써 작업의 정확성을 저해할 우려가 크다.

물론 ASCII 파일은 정보를 재처리하거나 상이한 프로그램에서 마음대로 불러다 사용하는 것이 매우 어려우며, 편집기로 데이터를 직접 확인할 수 있다고 하나 이는 원전인 데이터베이스에서 확인하는 것보다 신뢰성이 떨어지는 것이므로 결국 최종의 확실한 데이터가 아니므로 ASCII 파일의 사용은 가능한 한 피해야한다. 특히, ASCII 파일을 사용할 경우에는 파일의 수가 모듈의 수에 비례하여 피료하지만, 데이터베이스를 사용할 경우에는 기본적으로 하나의 데이터베이스만 필요하고 필요한 경우에는 메트릭스 구조로된 테이블로 모듈화하여 관리하는 것이 가능하다.

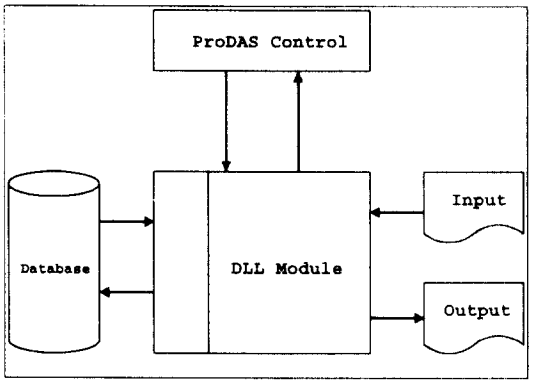

Fig. 1 A DLL module, under ProDAS's control, communicating with the database and/or I/O file system

요소 프로그램 모듈과 데이터베이스 시스템 사이의 연결을 확보하기 위하여는 우선 정보를 상용화된(따라서 사용이 용이하고 신뢰성이 있는) 데이터베이스 시스템에 일정한 형식으로 저장, 관리하는 것이 필요하다. 상용화된 관계형 데이터베이스 패키지(예를 들면, Oracle 또는 Access등)의 사용은 신뢰성 뿐아니라 두터운 사용자 층 때문에 사용법을 쉽게 익힐 수 있는 장점이 있다. 본 연구에서는 Microsoft사의 Access를 기본으로 사용하고 있으나 Oracle을 사용하여도 전혀 차이가 없다. Fig. 1은 ProDAS 제어체계의 조종을 받고 있는 한 요소 프로그램 모듈이 데이터베이스와 연결되어 있는 모양을 보여준다. (사용자가 원하면 전통적인 파일 입출력 방식과 연결하는 것도 가능하다.) 본 연구의 주된 내용은 각 프로그램 모듈과 데이터베이스 사이의 정보교류를 가능하게 해주는 인터페이스를 개발하는 것이다.

2. 데이터베이스 인터페이스 설계

2.1 설계 개요

단위 기관 또는 그룹에서 사용하는 데이터베이스는 한번 체계화되면 수많은 프로그램들이 여기에 연결되어 사용되기 때문에 데이터베이스 시스템을 바꾼다는 것은 업무의 연속성에 큰 영향을 줄 수 있다. 그러나 객체지향형 데이터베이스 기술이 개발되면서 지금까지 처리할 수 없었던 데이터를 다루는 것이 가능해져 조만간 기존의 데이터베이스를 객체지향형으로 교환할 필요성이 높아지고 있다. 따라서 Fig. 1에 보인 것과 같은 인터페이스를 개발할 때 기본적으로 향후의 객체지향 데이터베이스가 발표될 것에 대비한 인터페이스를 설계하는 것이 매우 중요한 조건이 된다[3]. 본 연구에서는 이를 대비하여 객체지향형 언어인 C++를 채택하였으며, 데이터베이스 관련 클래스를 설계하고 필요한 경우에는 항상 수정이 가능하도록 원시 프로그램을 작성하였다.

데이터베이스 시스템이 상용화되려면 컴퓨터 운용체계의 설계자와 데이터베이스의 개발자는 서로 협력하여 데이터베이스를 활용할 응용프로그램과 데이터베이스 자체 사이의 연결을 가능하게할 Driver를 우선적으로 제공한다. Microsoft사에서 데이터베이스의 프로토콜로 표준화하여 제공하는 ODBC(Open Database Connectivity) 또는 DAO (Data Access Objects)는 기존의 관계형 데이터베이스를 활용할 수 있도록 한 Driver의 예이다. 본 연구에서는 DAO를 써서 Access로 관리하는 데이터에 접근하여 이를 읽고 수정할 수 있는 인터페이스를 개발하였다. DAO Driver 이외에는 모든

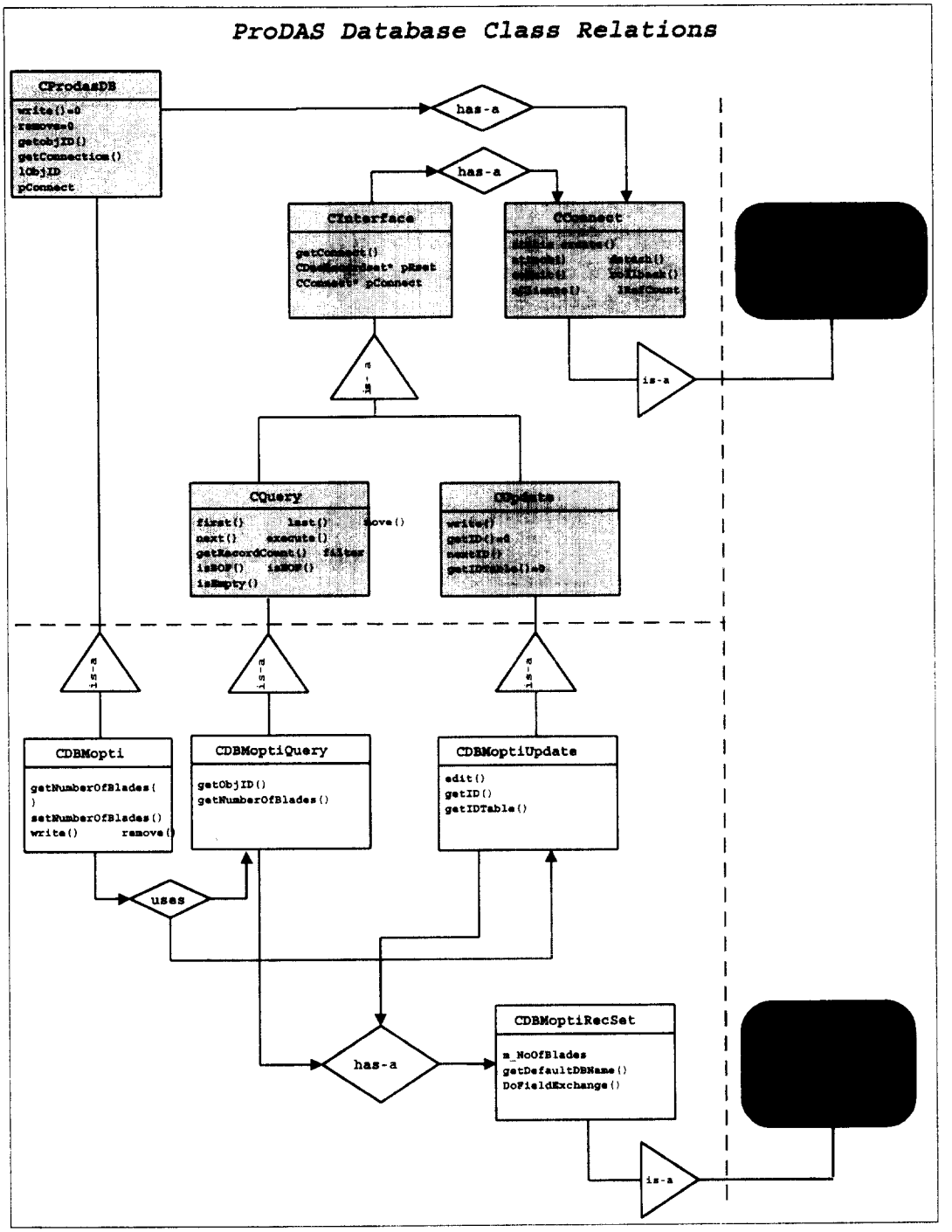

Fig. 2 ProDAS database class relations

것을 직접 다룰 수 있는 것이 본 연구의 장점이다.

Fig. 2는 ProDAS에 적용된 데이터베이스 인터페이스 클래스 사이의 관계와 이들 클래스가 세 개의 그룹으로 나뉘어 있음을 보여준다. 제일 우측의 CDaoDatabase 및 CDaoRecordset 클래스는 Microsoft사에서 제공하는 것으로 데이터베이스와 응용프로그램 사이를 연결시켜주거나 정보 통로를 만들어주는 역할을 하는 것으로 응용프로그램의 개발자는 이들 클래스의 멤버변수 및 멤버함수를 사용하는 방법을 알기만 하면된다. 좌측 상단의 CProdasDB, ..., CUpdate 등의 클래스는 ProDAS 패키지에 포함될 것으로 각 모듈이 데이터베이스에 연결되어 원하는 레코드셋트를 찾아가서 필요한 데이터를 읽거나 기록하는 기본적인 기능을 제공하는 클래스이다. 이들 클래스는 각 모듈의 내용과는 전혀 관계가 없으며, 데이터베이스의 특정한 테이블 또는 필드와 관계되어 있지 않다. 좌측 하단의 CDBMopti, ..., CDBMoptiRecSet 등의 클래스는 각 요소 프로그램 모듈이 직접 데이스베이스와 특정한 자료를 주고 받는데에 필요한 클래스로 이 부분은 각 모듈에 따라 작성되어야 하는 부분으로 ProDAS에 연결되는 데이터베이스를 활용하는 모든 모듈은 이와 유사한 클래스를 갖는다.

2.2 각 클래스의 기능 및 상호 관계

Fig. 2에서 CProdasDB 클래스는 ProDAS 데이터베이스 클래스 중에서 최상위에 있는 클래스로, 데이터베이스에 연결하는 멤버변수인 pConnect와 데이터베이스의 레코드를 가리키는 멤버변수인 lObjID를 가지고 있다. CConnect 클래스는 데이터베이스를 연결하기 위한 객체를 생성하고 데이터베이스를 여는(Open) 기능과 데이터베이스에 접속하거나(attach) 떨어질(detach) 때를 확인하며, 데이터베이스에 자료를 디스크에 기록하거나(commit) 이전의 디스크 기록 이후의 작업을 되돌리는(rollback) 기능을 갖는다.(attach 및 detach함수를 수정하면 다중사용자 환경에서 사용하는 것이 가능하다.) CInterface 클래스는 CConnect 클래스의 포인터인 pConnect와 CDaoRecordset 클래스의 포인터인 pRset를 멤버변수로 갖는다. 이 두 멤버변수는 CInterface 클래스로부터 유도되는 CQuery 클래스 또는 CUpdate 클래스에 상속되어 모든 데이터베이스에 공통으로 데이터베이스를 가리키거나 레코드를 가리키는데에 활용된다.

CInterface 클래스로부터 유도된 CQuery 클래스는 레코드셋트에서 질의 조건에 따라 레코드를 가리키는 포인터를 원하는 위치로 옮기는 데에 사용되며, CInterface로부터 유도된 CUpdate 클래스는 추상 클래스로 현재 위치에서 응용프로그램 모듈에서 수정된 데이터베이스에 기록하거나, 가장 뒤에 있는 레코드 다음 숫자를 알아내는 함수를 갖고 있다. 위의 5개의 클래스는 ProDAS 데이터베이스 체계의 가장 기본적인 클래스들로 데이터베이스의 제어에 필요한 모든 함수를 제공한다. 특히, 이 부분은 ProDAS에 연결되는 여러개의 모듈과 관계없이 존재한다.

모듈에 종속적인 클래스는 Fig. 2에 보인 바와 같이 CProdasDB로부터 유도되는 CDBMopti 클래스와 CQuery 및 CUpdate 클래스로부터 각각 유도되는 CDBMoptiQuery 및 CDBMoptiUpdate 클래스, CDaoRecordset 클래스로부터 유도된 CDBMoptiRecSet 클래스가 있다. 이중에서 가장 기본적인 클래스는 CProdasDB로부터 유도된 CDBMopti 클래스이다. 이 클래스는 CDBMoptiQuery 및 CDBMoptiUpdate 클래스를 사용할 수 있는 권한을 갖고 있어, 이 클래스의 객체를 생성하기만 하면 질의를 수행하거나 자료의 수정을 수행할 수 있고, 자기 자신의 멤버함수(예를 들어, getNumber-OfBlades() 또는 setNumberOfBlades())를 사용하여 원하는 자료를 읽거나 쓸 수 있다. 여기서 특히 관심을 끄는 클래스는 CDBMoptiRecSet으로 CDaoRecordset로부터 유도된 클래스로 데이터베이스 테이블에 들어있는 모든 필드의 값을 초기화하는 기능과 데이터베이스와 응용클래스 사이의 자료 교환의 창구 역할을 수행한다.

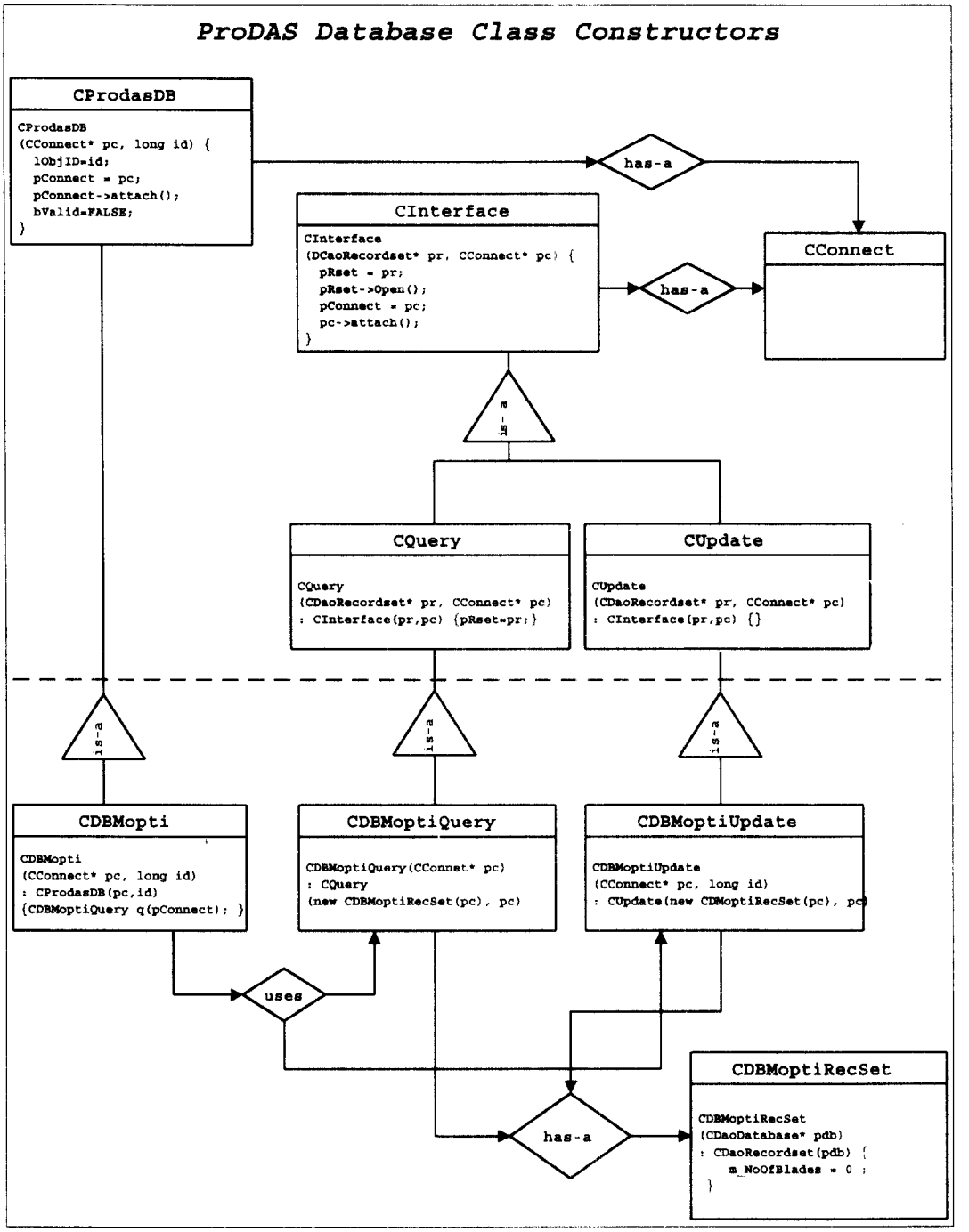

Fig. 3 ProDAS database class constructors

2.3 각 클래스의 생성자

클래스를 선언할 때, 각 클래스에는 기본적인 멤버변수 및 멤버함수와 함께 생성자가 함께 선언되고 정의된다. Fig. 3은 ProDAS 데이터베이스 시스템에서 사용되는 각 클래스의 생성자를 보여준다. Fig. 3의 CDBMopti 클래스 칸의 생성자를 보면, CDBMopti 클래스의 객체를 생성할 때, 컴파일러는 상위 클래스를 자동적으로 초기화하는 것을 우선 알 수 있으며, 이어서 생성자 함수의 몸체내에서 CDBMoptiQuery 클래스의 객체를 생성함을 볼 수 있다. 이어서 CDBMoptiQuery 클래스의 생성자를 호출하면, CQuery 클래스를 호출함을 볼 수 있고, CQuery 클래스는 CDBMopti-RecSet 클래스의 객체를 임시로 생성하고 데이터베이스에의 연결 포인터인 CConnect *pc를 CDBMoptiRecSet 클래스의 매개변수인 CDaoData-base *pdb로 치환하고 이어서 CDaoRecoreset 클래스에 연결한다. 지금까지의 연속 생성 과정을 통하여 CDBMopti 클래스의 객체는 CQuery 클래스의 첫 번째 매개변수를 통해 CDaoRecordset 클래스와 연결이 되었으며, 이어서 CQuery 클래스의 생성자를 통해 CInterface 클래스를 초기화하여 여기까지 전달된 CDaoRecordset *pr과 CConnect *pc를 가지고, CDBMopti 클래스의 객체 뿐만아니라 이하 CUpdate 클래스로부터 유도된 클래스의 객체가 생성될 때에도 함께 사용될 CDaoRecordset* pRset과 CConnect* pConnect 등 두 개의 전용변수를 정의한다.

2.4 데이터베이스와 응용프로그램의 정보 전달

데이터베이스 인터페이스 프로그램이 완성되면 DBMoptiQuery.cpp 및 DVMoptiRecSet.cpp 등과 같은 파일을 갖추게 된다. Fig. 4는 이렇게 생성된 코드의 일부를 보여준다. Fig. 4는 우선 CDBMoptiQuery 클래스의 배열로 데이터베이스의 필드와 대응되는 질의 테이블이 만들어짐을 보여준다. 여기에서는 테이블의 명칭 "Mopti"와 특정한 필드인 [NoOfBlades]만을 보이기로 한다. 이어서 getNumberOfBlades()함수가 구현된 모양을 보여준다. 이 함수의 역할은 레코드셋을 가리키고 있는 pRset의 변수값 m_NoOfBlades를 되돌리는 것이다. m_NoOfBlades는 DBMoptiRecSet프로그램에서 구현된 DoFieldExchange()함수를 통하여 데이터베이스의 [NoOfBlades]와 정보를 교환한다.

```
// DBMoptiQuery.cpp
  . . . . . .
char* CDBMoptiQuery::queryTable[CDBMoptiQuery::NFIELDS+1] =
        {"Mopti",. . .,"[NoOfBlades]",. . .};
  . . . . . .
int CDBMoptiQuery::getNumberOfBlades() const
{
  return ((CDBMoptiRecSet *)pRset)->m_nblade;
}
  . . . . . .

// DBMptiRecSet.cpp : implementation file
  . . . . . .
CDBMoptiRecSet::CDBMoptiRecSet(CDaoDatabase* pdb)
 : CDaoRecordset(pdb)
{
  //{{AFX_FIELD_INIT(CDBMoptiRecSet)
  m_NoOfBlades = 0;
  . . . . . .
  //}}AFX_FIELD_INIT
  m_nDefaultType = dbOpenDynaset;
  //m_bCheckCacheForDirtyFields = FALSE;
}
void CDBMoptiRecSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
  . . . . . .
  DFX_Long(pFX, _T("[NoOfBlades]"), m_NoOfBlades);
  . . . . . .
}
```

Fig. 4 Segments of Query and Recordset implementation files

2.5 데이터베이스 객체의 응용프로그램

이상의 과정을 통하여 데이터베이스의 클래스 체계를 완성하면 응용프로그램을 사용하여 객체를 생성하는 코드 작성만 남는다. Fig. 5는 ProDAS의 한 DLL 모듈이 데이터베이스와 연결되어 자료를 추출하고 기록할 수 있는 방법을 보여주는 코드 일부의 예를 보여준다. 이 예에서 볼 수 있듯이 전처리 변수 _ProDASDBase를 사용하여 원시 프로그램을 원하는 부분을 골라서 컴파일할 수 있음을 우선 관찰할 수 있다. 이로서 기존의 원시

프로그램을 거의 그대로 활용함으로써 프로그램의 개발과정을 쉽게할 수 있음을 알 수 있다. 우선, 데이터베이스를 연결하고 CDBMopti 클래스의 객체 mopti를 생성한 후에, InputFromDBFile()이라는 함수를 호출함으로써 데이터베이스로부터의 입력을 수행할 수 있다. 이어서 기존의 응용프로그램에서 사용중인 변수 nvel은 mopti.getNumberOfVelocity()함수를 통하여 치환하기만 하면된다. 다른 모든 입력 변수는 이와 유사한 방법을 써서 데이터베이스의 값을 추출하여 사용할 수 있다. 응용프로그램에서 계산된 값을 데이터베이스에 저장하는 방법도 이와 유사하게 OutputToDBFile() 함수를 통하여 CDBMopti객체를 생성하여 값을 저장할 수 있다. 이상에서 살펴본 바와 같이 데이터베이스용 클래스가 완성되면 사용하는 방법은 매우 간단함을 알 수 있다.

```
#if !defined(_ProDASDbase)
#define _ProDASDbase
#endif
.....
extern "C" int WINAPI MOpti(CListBox FAR* mDlg, long id)
{
#ifdef _ProDASDbase
  CConnect *pc = CConnect::create("WWProDASWWProDBase.mdb");
  CDBMopti mopti(pc, id);
  InputFromDBFile(mopti);
#else
  .... File input statements .....
#endif _ProDASDase
  .... Module DLL routines .....
#ifdef _ProDASDBase
  OutputToDBFile(mopti);
  pc->detach();
#endif
  ..... File output statements .....
}
void InputFromDBFile(CDBMopti &mopti)
{
  ident = mopti.getIdent();
  nvel = mopti.getNumberOfVelocities();
  igivend = mopti.getiGivenDiameter();
    .....
  for(n=0; n<nvel; n++) {
    sus[n] = mopti.getShipUs(n);                    // SI
    .....
  }
    .....
}
void OutputToDBFile(CDBMopti &mopti)
{
  mopti.setPropDiameter(ddiam);
  mopti.write();
}
```

Fig. 5 Segments of a sample DLL code

3. ProDAS 제어체계의 수정

기존의 ProDAS 패키지는 데이터베이스를 활용하지 않고 ASCII 파일 형태로 입출력을 수행하였다. 따라서, 데이터베이스와의 연결을 가능하게 하기 위하여 Windows 제어체계를 수정할 필요가 있다. Fig. 6는 대화상자에 데이터베이스를 선정하는 선택단추가 추가되어있음을 보여주며, 여기서 MOpti.mdb 단추는 설계, 해석 프로그램 패키지가 기존의 데이터베이스를 사용할 것인지 또는 새로운 데이터베이스를 사용할 것인지를 결정할 수 있는 데이터베이스 인터페이스를 뜨게 한다. 그 아래의 Edit Box는 현재 설계 또는 해석 중인 프로펠러의 고유번호(IPropID)를 묻는다. 하나의 프로펠러를 설계할 때, 여러개의 상이한 모듈을 사용하므로 각 프로그램 사이에 공유되는 특정한 프로펠러 고유번호를 사용하는 것이 필요하기 때문이다. 이러한 제어체계를 통하여 설계, 해석 작업을 새로 시작하거나 다시 작업하는 경우 신속히 해당 module에 각 data가 전해지도록 하고, 장차 데이터베이스의 레코드가 충분히 커지면 여러 가지 통계해석이 가능하고, 이들 통계해석의 결과를 활용하여 새로운 프로펠러를 설계할 수 있음을 보인다.

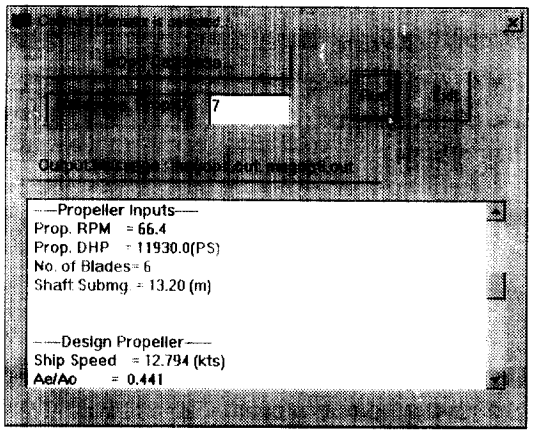

Fig. 6 A dialog box menu showing controls related to ProDAS Database

4. 예제 응용 결과 및 토의

Fig. 1에서 살펴보았듯이 ProDAS의 한 모듈과 데이터베이스는 서로 따로 존재하며, 본 연구에서 설명한 인터페이스를 통하여 정보를 주고 받는다. ProDAS의 각 모듈에서 사용될 입력자료는 Access 데이터베이스에 직접 입력되며, Fig. 7은 자료입력 화면을 보여준다. 이렇게 입력된 자료는

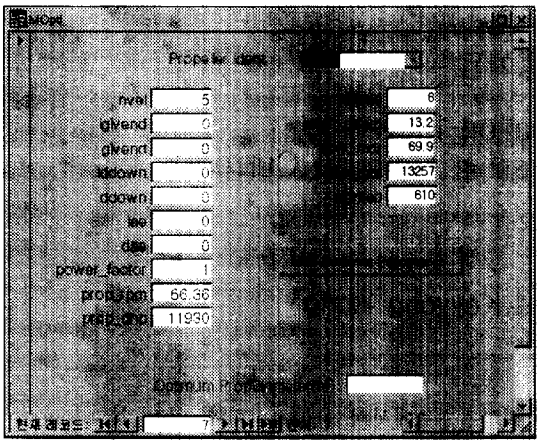

Fig. 7 Typical view of the database dialog box

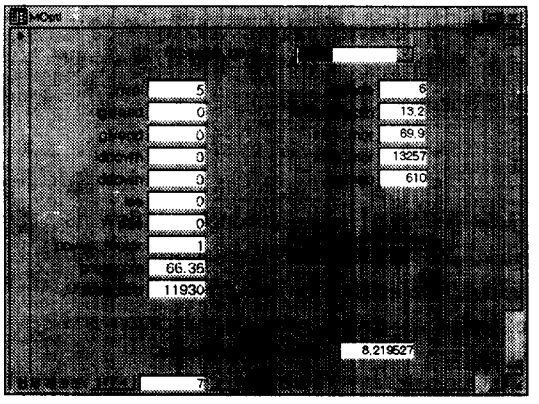

Fig. 8 Typical view of the database showing the data updated directly from the application program

인터페이스를 통하여 모듈에서의 직접 계산에 사용되며, Fig. 6에서 보인 바와 같이 ASCII 형식의 출력을 내보내거나 다른 모듈에서의 계산을 위해 다시 데이터베이스에 저장되거나 한다. Fig. 8은 Optimum Diameter값이 계산이 끝난 후에 update 되었음을 보여준다. 입출력의 예에서 보았듯이 데이터베이스를 직접 사용하는 것은 매우 간단함을 알 수 있다. 자료의 관리는 데이터베이스 소프트웨어 자체의 신뢰성만큼 정확하게 관리될 수 있음은 물론이다.

5. 결언

1) 종합적인 패키지에서 각각의 요소 프로그램 모듈과 데이터베이스사이의 정보 교류를 원활히 수행할 수 있는 인터페이스 클래스를 설계하고 코드를 완성하였다.
2) 응용프로그램과 데이터베이스 사이의 정보 교류가 원활히 수행됨을 확인하였다.
3) 인터페이스는 데이터베이스의 구조가 객체지향형으로 바뀌더라도 쉽게 변환이 가능하도록 설계, 개발되었다.
4) 데이터베이스, 요소 프로그램, 인터페이스들이 각각 모듈화되었기 때문에 유지, 관리와 확장이 용이할 것이다.

참 고 문 헌

[1] 정인숙, 이창섭, 임효관, "GUI를 이용한 프로펠러 설계법," 대한조선학회, 1996년 춘계연구발표회, 삼성중공업, 1996, pp. 220-223.
[2] 이창섭, 임효관, 정인숙, 조충호, "PC를 이용한 설계 프로그램 작성 기법," 대한조선학회 선박설계연구회 1997년도 동계 연구발표회, 2.20-2.21, 1997, pp. 181-197