객체 재사용성 향상을 위한 레거시 시스템 인터페이스기반 객체추출 기법

이창목[0], 최성만, 유철중, 장옥배
전북대학교 컴퓨터과학과
{cmlee[0], sm3099, cjyoo, okjang}@chonbuk.ac.kr

A Technique of Object Extraction Based on the Legacy System Interface for the Improvement of Object Reusability

Chang-Mog Lee[0], Seong-Man Choi, Cheol-Jung Yoo, Ok-Bae Chang
Dept. of Computer Science, Chonbuk National University

요 약

본 연구는 레거시 시스템의 인터페이스 정보로부터 의미 있는 정보를 파악하여 새로운 시스템에 통합될 수 있도록 하기 위한 기존 레거시 시스템의 인터페이스에 기반한 객체추출 기법을 제안한다. 본 논문에서 제안하는 객체추출 기법은 인터페이스 사용사례 분석 단계, 인터페이스 객체 분할 단계, 객체구조 모델링 단계, 객체 모델 통합 단계 등 4단계로 구성되어 있다. 인터페이스 사용사례 분석 단계는 인터페이스 구조, 레거시 시스템과 사용자간의 상호작용 정보를 획득하는 단계이다. 인터페이스 객체분할 단계는 인터페이스 정보를 의미 있는 필드들로 구분하는 단계이며, 객체구조 모델링 단계는 인터페이스 객체들간의 구조적 관계와 협력 관계를 파악하여 모델링하는 단계이다. 마지막으로 객체 모델 통합 단계는 객체 단위의 단위 모델들을 통합하여 추상화된 정보를 포함한 상위 수준의 통합 모델을 유도하는 단계다. 객체추출 기법에 의해 생성된 객체 통합 모델은 역공학 기술자들의 레거시 시스템 이해와 레거시 시스템의 정보를 새로운 시스템에 적용하는데 있어 효율성을 극대화할 수 있다.

1. 서 론

객체지향 패러다임이 제시된 이래 오늘날 대부분의 시스템을 개발하는데 객체지향 패러다임이 폭 넓게 적용되어가고 있는 추세이다. 그 이유는 객체지향 패러다임이 갖는 재사용성과 새로운 시스템에 자연스럽게 통합되어질 수 있는 유연함 때문이다[1]. 그러나 아직도 대부분의 기업에서는 비 객체지향적 이거나 혹은 객체지향 언어를 사용하여 시스템을 개발하였으나 객체지향 개념이 정확히 적용되지 않은 레거시 시스템을 그대로 사용하고 있다. 그러나 이러한 구시대적으로 개발된 시스템은 급변하는 업무 환경에 적합하게 대처할 수 없는 경우가 대부분이다[2, 3].

인터페이스는 최종 사용자를 위한 사용자 인터페이스이다. 다시 말하면, 애플리케이션을 사용하는데 있어 최종 사용자로 하여금 가장 일반적으로 사용되는 공통된 수단이다. 애플리케이션 시스템 대부분의 지식은 인터페이스와 그 인터페이스를 통해서 시스템과 대화하는 상호작용(interaction)에 의해 정보가 발생한다. 따라서 본 기법은 최종 사용자와 시스템간의 상호작용 과정을 통하여 발생된 인터페이스 정보를 자연스럽게 분류 가능한 것이다. 본 논문의 구성은 다음과 같다. 먼저, 2장에서는 객체추출 기법을 3장에서는 에이전트기반 정보수집 예를 설명하고 4장에서는 다른 기존의 역공학 방법론과 비교를 5장에서는 결론 및 향후 연구과제에 대하여 설명한다.

2. 객체추출 기법

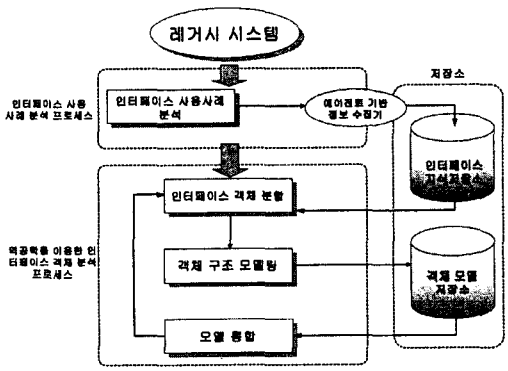

그림 1 객체추출 기법의 구조도

첫째, 인터페이스 사용사례 분석 단계는 레거시 애플리케이션의 인터페이스 필드 정보와 인터페이스를 통해 사용자가 시스템과 상호 작용한 정보를 얻기 위한 단계이다. 사용자가 레거시 시스템의 인터페이스를 통한 상호작용의 예는 입력 필드에 어떠한 데이터를 입력하는 행위, 입력한 데이터를 처리하기 위해 처리 시스템에 보내기 위한 이벤트 발생 행위, 이벤트에 의해 발생되어진 입력 데이터를 처리하는 연산행위, 연산행위를 통해 처리된 데이터를 사용자 인터페이스에 다시 반환하는 행위, 데이터베이스로부터 데이터를 인터페이스에 넘겨주는 행위 등 여러 가지의 예가 있다. 이러한 상호작용 정보와 인터페이스에 관한 정보가 에이전트기반 정보 수집

기에 의해 자동 수집된다. 에이전트 기반 정보 수집기에 의해 얻어질 수 있는 인터페이스 구성 정보유형은 다음 네 가지의 유형이다.

① 인터페이스 영역유형은 인터페이스의 영역별 기능을 나타내는 유형이다. 따라서 데이터의 항목을 나타낼 수 있는 항목별 필드, 항목의 수를 카운트 할 수 있는 집계 필드, 그리드 영역, 이벤트 컨트롤, 조건부 입력 영역 등으로 세부 분류가 가능하다.

② 입력지원 컨트롤 유형은 사용자가 인터페이스에 데이터를 입력하는 방법에 여러 유형이 있을 수 있는데 상황에 따라 사용자가 편리하게 데이터를 인터페이스에 입력할 수 있도록 한다.

③ 인터페이스의 필드를 편집할 수 있는 인터페이스 편집필드가 있다. 키 입력 필드, 데이터 입력필드, 조회와 수정 둘 모두 가능한 필드, 조회만 가능한 필드, 이벤트 발생 필드, 시스템 관리 필드의 유형이 있다.

④ 처리 적용형태에 따른 유형이 있다. 인터페이스 상에서 일어날 수 있는 처리 적용형태는 앞서 설명한 조회, 등록, 수정, 삭제의 크게 4가지가 있다. 이는 다시 인터페이스 상에서 이루어 질 수 있는 처리 건별로 단일건 처리와 다수건 처리로 구분될 수 있으며 이는 또다시 데이터베이스를 기준으로 해서 다른 테이블과의 조인 또는 무조인으로 구분될 수 있다.

```
DB  DBList, Sgl_DBList; // DBList는 모든 Sgl_DBList들에 대한 정보를
갖고 있는 연결리스트
while(!All_Interface_exit()) // 모든 인터페이스에 대하여 반복
{
   getSgl_Interface(); // 단일 인터페이스 획득
   while(!exit)
   {
    getInter_User_System(); // 에이전트 시스템을 통한 사용자와 시스템간의
상호작용 정보 획득
     Sgl_DBList = saveAll_Interface_Inform(); // 획득된 단일 인터페이스 정보를 저장
   } // end while
   addList(DBList, Sgl_DBList);  // 단일 DBList를 DBList에 추가
} // end while
```

그림 2 인터페이스 사용사례 분석 알고리즘

둘째, 인터페이스 객체분할 단계는 전 단계의 에이전트 프로그램에 의해 수집되었던 인터페이스에 관한 정보를 이용한다. 인터페이스에 관한 정보가 저장되어있는 인터페이스 지식 저장소로부터 인터페이스에 관한 다양한 분류 정보를 입력받는다. 그런 다음 인터페이스에 나타나있는 다양한 필드들의 정보를 영역별(항목, 집계, 그리드, 이벤트, 조건부 영역 등) 및 입력지원 컨트롤(콤보박스, 서브인터페이스, 라디오 버튼, 체크버튼 등) 그리고 인터페이스 편집필드 유형(키 입력 필드, 데이터 입력필드, 조회 및 수정 필드, 조회 필드, 이벤트 발생 필드, 시스템 관리 필드) 등의 인터페이스 정보의 유형에 따라서 인터페이스에 관한 지식을 의미 있는 정보 단위 객체로 분할한다. 그림 3은 인터페이스 객체 분할 알고리즘을 보여준다.

셋째, 객체 구조 모델링 단계는 이전 과정의 단계에서 나타난 결과로부터 객체들을 파악하는 단계이다. 또한 객체들 간의 초기 수준의 구조적(structural) 관계와 협력(collaboration)관계가 유도되는 단계이다. 이 단계에서 나타나는 결과물은 유도된 객체의 객체 명, 객체 속성, 그리고 구조적 관계로 구성된 객체구조 모델이 형성된다. 또한 객체구조 모델을 나태내기 위해 사용한 표기법으로는 현재 모델링 언어의 표준인 UML을 이용하여 나타낸다. 그림 4는 객체구조 모델링 알고리즘을 보여준다.

```
while (DBList != NULL) // DBList: 에이전트에 의해 획득된 모든 정보 리스트
{
  Object  InterOb, Sgl_InterOb; // 인터페이스 객체
  Object  ComplexOb, Sgl_ComplexOb; // 복합 객체
  Object  DBaccessOb, Sgl_DBaccessOb; // DB 접근 객체
  Object  ExtraOb, Sgl_ExtraOb; // 부가적인 객체
  Sgl_InterOb = createInterfaceObject(DBList);  // 단일 인터페이스 객체 생성
  giveNameToObject(Sgl_InterOb.name);
  // 인터페이스 이름을 이용하여 객체명 부여
  givePropertyToField(Sgl_InterOb.field); // 인터페이스 필드에 속성 할당
  addList(InterOb, Sgl_InterOb); // 인터페이스 객체 리스트에 추가
  while (Sgl_InterOb.field != NULL) // 단일 인터페이스내의 모든 필드 검색
  {
   if (isComplexField(Sgl_InterOb.field)) // 복합 필드 검색
     {
       Sgl_ComplexOb = createObject(Sgl_InterOb.field);
       // 복합 필드를 복합 객체로 유도
       giveNameToObject(Sgl_ComplexOb.name); // 객체명 부여
       givePropertyToField(Sgl_ComplexOb); // 속성 부여
       addList(ComplexOb, Sgl_ComplexOb); // 객체 리스트에 추가
     } // end if
    if (isDBaccess(Sgl_InterOb.field)) // DB 접근 필드 검색
    {
      Sgl_DBaccessOb = createObject(Sgl_InterOb.field);
      // DB 접근 필드를 DB 접근 객체로 유도
      giveNameToObject(Sgl_DBaccessOb.name); // 객체명 부여
      givePropertyToField(Sgl_DBaccessOb.field); // 속성 부여
      addList(DBaccessOb, Sgl_DBaccessOb); // 객체 리스트에 추가
    } // end if
    else if (checkObject(Sgl_InterOb.field)) // 부가적인 필드 검색
    {
        Sgl_ExtraOb = createObject(Sgl_InterOb.field);
        // 부가적인 필드를 부가적인 객체로 유도
        giveNameToObject(Sgl_ExtraOb.name); // 객체명 부여
        givePropertyToField(Sgl_ExtraOb); // 속성 부여
        addList(ExtraOb, Sgl_ExtraOb); // 객체 리스트에 추가
    } // end if
    Sgl_InterOb.field = Sgl_InterOb.field.next; // 다음 필드로 이동
  } // end while
  DBList = DBList.next; // 다음 단일 인터페이스로 이동
} // end while
```

그림 3 인터페이스 객체 분할 알고리즘

```
while (InterOb != NULL) // 모든 인터페이스 검색
{
  while (InterOb.ExtraOb != NULL) // 모든 집계 필드 후보 검색
  {
    if (isCompute_field(InterOb.ExtraOb)) // 집계 필드 검증
    {
       Compute_Inform = callAgentSystem(InterOb.ExtraOb);
       // 에이전트로 부터 집계 필드와 관계된 모든 정보 획득
       while (Compute_Inform != NULL)
       {
         operator = searchOperator(Compute_Inform); // 최소단위 연산자 검색
         operand = searchOperand(Compute_Inform); // 최소단위 피연산자 검색
         variable = initVariable(operand); // 변수 설정
         relationOb = searchObject(InterOb, operand);
         // 피연산자와 관계된 객체 검색
         setRelation(variable, relationOb); // 객체와 변수와의 관계 설정
         method = createMethod(variable, operator);
         // 연산자와 피연산자와의 계산식 생성
         Compute_Inform = Compute_Inform.next;
         // 집계 필드에 관계된 다음 정보로 이동
       } // end while
       setMethodToObject(method, InterOb.ExtraOb);
       // 집계 필드 객체에 메소드 할당
       InterOb.ExtraOb = Sgl_ExtraOb.next; // 다음 집계 필드 후보로 이동
    } // end if
  } // end while
  InformOb = InformOb.next; // 다음 인터페이스로 이동
} // end while
```

그림 4 객체 구조 모델링 알고리즘

넷째, 모델 통합 단계이다. 모델 통합 단계의 목적은 전 단계의 결과물인 객체정보를 나타내는 단위 모델들을 통합하여 보다 상위 수준의 통합 모델을 제시하고자 함이다. 결과적으로, 객체추출 기법의 과정을 통해 얻어지는 최종 결과물은 객체 구조 모델이다. 그림 5는 모델 통합 알고리즘을 보여준다.

```
model = NULL; // model 변수의 초기값 설정
alter_model = NULL; // alter_model 변수의 초기값 설정
while (!isSuitableModel(model)) // 모델 적합성 판단
{
  model = selectStruc(InterOb); // 전체 인터페이스 객체 구조 모델 선택
  InterOb = InterOb.next; // 다음 인터페이스로 이동
  while (InterOb != NULL) // 모든 인터페이스 객체 검색
  {
    alter_model = selectStruc(InterOb); // 이동한 인터페이스의 객체 구조 모델 선택
    if (isDifferModel(model, alter_model)) // 두 모델의 유사성 판단
    {
      mergeSameOb(model, alter_model); // 공통 객체를 기반으로 모델 통합
      mergeByRelation(model, alter_model); // 관계있는 모델 통합
      solveAmbiguousName(model, alter_model); // 모호한 이름 해결
      solveAmbiguousStruc(model, alter_model); // 모호한 구조 해결
      model = saveToAll(); // 통합된 모델 저장
    } // end if
    InterOb = InterOb.next; // 다음 인터페이스 지정
  } // end while
} // end while
```

그림 5 모델 통합 알고리즘

3. 에이전트 기반 정보 수집기의 정보 수집 예

레거시 시스템으로부터 객체 정보를 얻기 위한 과정은 레거시 시스템의 인터페이스 환경이 CUI와 GUI(본 논문에서는 마이크로소프트사의 운영체제를 표준 플랫폼으로 가정했음)인 형태가 있다. 첫째, CUI 환경에서는 VGA 그래픽카드의 경우 B000:B800 은 텍스트 디스플레이를 위한 비디오 프레임 버퍼 공간이다. 따라서 CUI 에서 실행되는 에이전트를 램상주 프로그램으로 작성하고, 특정 이벤트(일반적으로, 단축키를 누르면) 발생시 프레임버퍼를 스캔한다. 이렇게 하면 화면 정보를 그대로 불러올 수 있다. B000:B800 메모리는 <속성, 문자코드> 의 2바이트 정보가 반복되어 있기 때문에, 속성은 배경색, 전경색, 강조, 깜빡임 등의 속성이고, 문자코드는 씌어지는 문자의 코드 값이다. 예를 들어 화면에 "date:[ ]" 라는 형식으로 되어 있다면 date를 변수로 잡을 수 있으며, [ ] 같은 문자나 색상이 달라지는 것을 통해서 입력을 위한 공간을 예측할 수 있다.

둘째, GUI 환경에서는 GUI에 등록되어있는 객체 정보를 얻어낼 수 있는 소프트웨어를 이용하여 인터페이스 정보를 얻어낼 수 있다. 마이크로소프트사의 윈도우 환경이라면 "SPY++"라는 소프트웨어를 이용하여 인터페이스 객체정보를 획득할 수 있으므로 에이전트는 "SPY++"를 통해 획득된 정보를 텍스트 파일로 저장하여 이를 토대로 객체정보를 추출한다. 그림 6은 GUI 환경에서 실행되는 애플리케이션의 한 종류이다. 여기에서는 "Serial Number Generation and Add"라는 버튼에 해당되는 정보를 나타내주고 있다.

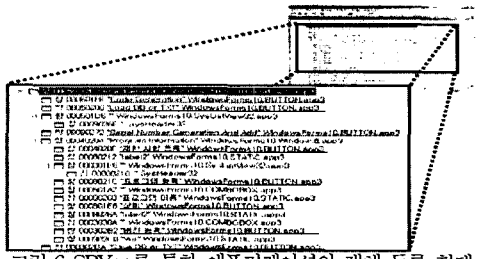

그림 6 SPY++를 통한 애플리케이션의 객체 등록 형태

4. 기존 역공학 방법론과의 비교

표 1 기존 역공학 방법론과의 비교

방법론 / 항목	Chiang	RE2	FORE	본 기법
시스템 관점	데이터	프로세스	객체	객체
목적	물리적 DB에서 의미 데이터 복구	레거시 시스템으로부터 재사용 모듈 생성	레거시 시스템으로부터 객체 구조 유도	· 레거시 시스템으로부터 객체구조 유도 · 통합 모델 추출
입력 데이터	물리적 DB 스키마	레거시 시스템의 모든 자료	입력화면 양식과 사용자의 상호작용 정보	· 레거시 시스템의 인터페이스 정보 · 레거시 시스템/사용자간의 상호작용정보
결과 모델	EER 모델	특별한 모델 없음	ECRC를 이용한 객체 모형	· UML에 기반한 객체(클래스)다이어그램 · 통합 모델

5. 결론 및 향후 연구

본 논문의 연구 결과로는 첫째, 입력자료는 사용자/시스템간의 가장 기본적으로 일어나는 인터페이스에 기반한 상호작용 정보를 활용한 역공학 연구이고 둘째, 인터페이스에 관한 지식을 지식 저장소에 저장할 수 있는 방법을 알고리즘화 된 단계로 제시하였으며, 셋째, 데이터 및 프로세스를 동시에 다루고 있다는 것이다.

향후 연구사항으로는 에이전트 기반 정보 수집기를 완성해야 하며, 객체추출 기법에서 보여준 객체 분석 과정을 자동화할 필요성이 있다.

6. 참고문헌

[1] H. Lee and Ch. Yoo, "A Form Driven Object-oriented Reverse Engineering Methodology", Information Systems, Vol. 25, pp. 235-259, May, 2000.

[2] Aiken, P., A. Muntz, R. Richards, "A framework for reverse engineering DoD legacy information systems", Proceedings Working Conference on Reverse Engineering, Baltimore, Maryland, May, pp. 180-191, 1993.

[3] Umar, A., Application Engineering Building Web-Based Applications and Dealing with Legacies, Prentice Hall, 1997.