제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

레거시 래핑 컴포넌트 생성 및 검증

이 문 수, 김 동 관
한국전자통신연구원(ETRI)
e-mail: mslee@etri.re.kr, dgkim@etri.re.kr

Generation and Verification of the Legacy Wrapping Component

Moon Soo Lee*, Dong-Kwan Kim**
*Dept. of ETRI-Computer & Software Technology Lab.
**Dept. of ETRI-Computer & Software Technology Lab.

요 약

레거시 시스템은 수년간 기업에서 많은 노력과 비용을 들여 개발되어 왔으며 현재는 기업의 중요한 자산으로 여겨지고 있다. 하지만 수많은 수정을 거치면서 시스템은 점차 비구조화 되어지고 그에 따른 문서화 작업이 제대로 이루어지지 않았으며, 과거의 중앙 집중적인 메인 프레임환경을 웹과 같은 분산 환경으로 이전하고자 하는 비즈니스 요구사항이 점차 증대되고 있다. 하지만 기존 시스템을 완전히 배제한 새로운 시스템을 구현하고 안정성을 테스팅하는 것은 하나의 큰 도전이 된다. 본 논문에서는 IBM 메임 프레임에서 운용되고 있는 레거시 COBOL 시스템을 연계하는데 있어서 보다 빠르고 안정성이 있는 컴포넌트 래핑 기술을 이용하여 엔터프라이즈 자바 빈(EJB)으로 생성하는 기법을 소개한다. 이에 따라 생성된 EJB를 검증하기 위한 기법을 제안한다.

1. 서론

레거시 시스템은 기업의 중요한 자산으로 여겨지며 이들은 COBOL 과 같은 프로시져 언어를 구현되었다. 하지만 기업의 추가 요구 사항에 따른 수정과 보완 작업이 계속되었지만 문서화 작업은 완전하게 이루어지지 못했다. 시스템 문서화와 프로그램 전문가의 부족은 점차 시스템의 유지 보수 비용이 증가하고 있다. 따라서 웹을 통한 기업의 서비스 환경 구축과 시스템의 유지 보수 비용을 줄이기 위해서는 레거시 시스템의 진화는 필연적이다. 레거시 시스템을 새로운 소프트웨어 환경으로 진화하기 위해서는 기존 시스템으로부터 재 사용할 수 있는 기능(Functionality)들을 식별하고 이를 컴포넌트 기반의 시스템에서 사용할 수 있는 새로운 컴포넌트로 패키징하고 테스팅 해 줄 수 있는 자동화 된 도구가 요구된다.

레거시 시스템의 진화에는 시스템 전반을 수정하는 경우와 부분적으로 확장이 필요하거나 재사용하고자 하는 모듈만을 진화 시키는 경우가 있다. 기존 시스템으로부터 재사용 모듈을 식별하는 것은 기업 업무에서 중요한 비즈니스 로직(Business Logic)을 찾는 것과 유사하다. 하지만 시스템의 프로그램 소스코드에서 이러한 비즈니스 로직을 식별하고 이해하는 일은 쉽지 않다. Ning[5]은 집중(Focusing), 선별(Selection), 재구조화(Factoring)단계들로 구성된 프로그램 분할(Program Segmentation) 기법을 사용하여 레거시 시스템으로부터 재사용 할 수 있는 기능성(Functional) 컴포넌트를 추출하였다. Hung[8]은 영역 변수(Domain Variable) 식별 단계, 슬라이싱 판별 기준 식별 단계, 일반화된 프로그램 슬라이싱 단계등 3 단계들로 나누었다. 그리고 각 단계마다 heuristic 한 기법들을 제시하였다.

한편, 식별된 비즈니스 로직은 컴포넌트 래핑 기법을 통하여 하나의 컴포넌트로 생성된다. 생성된 컴포넌트가 그 로직과 동일한지 검증할 수 있어야 한다.

본 논문에서는 비즈니스 로직과 관련성을 가지고 있는 데이터를 찾기 위해서 변수 분류(Variable Classification)법과 슬라이싱 판별 기준를 통하여 재사용 하고자 하는 워크플로워를 식별하고 컴포넌트 래핑을 이용하여 EJB 컴포넌트를 생성한다. 그리고 제안된 식별 기법을 이용하여 식별된 워크플로워에서 생성된 컴포넌트를 검증하기 위한 자동화된 테스팅 기법에 대한 내용을 다루게 된다.

본 논문의 구성은 다음과 같다. 2 장에서는 재사용 컴포넌트 식별에 대한 내용을 설명하였다. 3 창에서는 식별된 워크플로워를 컴포넌트 래핑으로 캡슐화에 대해 설명한다. 4 장에서는 생성된 래핑 컴포넌트를 테스팅하기 위한 방법을 제시하고, 마지막 5 장은 결론과

추후과제에 대해 논의할 것이다.

2. 레거시 컴포넌트 식별

레거시 컴포넌트 식별은 우선 프로그램에서 사용되어진 변수들의 연관성을 고려하여 각 변수를 그룹화한다. 그리고 사용자가 재사용하고자 하는 레거시 컴포넌트를 식별한다. 레거시 컴포넌트는 재 사용성이 높은 비즈니스 로직을 포함하고 있는 하나의 워크플로워를 식별하는 것이다. 워크플로워 식별은 제안된 슬라이싱 판별 기준을 통해 찾아내게 된다.

2.1 변수 분류(Variable Classification)

변수 분류는 미리 정의된 분류 그룹에 따라 프로그램에서 사용된 변수들을 그룹화하는 것이다. Kawabe[1,2]는 Y2K 문제를 해결하기 위해 COBOL 의 프로시져와 구조적인 분석을 통하여 변수 유형을 그룹화 하였다. Joiner[3]은 영역변수, 입력 변수, 프로그램 변수, 국지적 변수, 전역변수, 출력변수, 제약조건 변수, 제어변수와 같이 8 개의 변수 형태를 정의하고 그룹화하였다.

일반적으로 비즈니스 로직은 임의의 입력 변수 집합(x)과 출력 변수 집합(y)으로 표현되는 하나의 방정식으로 표현할 수 있다.

$$\sum y_j = f(\sum x_i)$$

재사용하기 위한 기능을 포함하고 있는 비즈니스 로직을 찾기 위해서는 그 로직을 표현하기 위한 영역변수와 밀접한 관계를 가지는 변수를 알아내는 것이 중요하다. 중요한 영역변수는 입력이나 출력 변수들과 밀접한 관련성을 가지게 된다.

COBOL 프로그램에서 입출력에 관련된 정보를 가지고 있는 부분이 Map 파일이 된다. 이 Map 파일은 프로그램의 로직과 사용자 인터페이스를 분리시켜 주는 기능을 한다. 그림 1 은 Map 파일의 예를 보여준다.

```
KCB0203   DFHMSD TYPE=&SYSPARM,LANG=COBOL,MODE=INOUT,TIOAPFX=YES,*
          CTRL=(FREEKB,FRSET),STORAGE=AUTO,EXTATT=YES,SOSI=YES
LTER110   DFHMDI SIZE=(24,80),LINE=01,COLUMN=01
          DFHMDF POS=(01,20),LENGTH=42,ATTRB=(BRT,PROT),          *
          INITIAL='터 미 널 관 리 화 일 내 용'
                              ⋮
KEYGRP1   DFHMDF POS=(03,20),LENGTH=04
          DFHMDF POS=(03,25),LENGTH=01,INITIAL=':'
KEYGRP2   DFHMDF POS=(03,27),LENGTH=08,PICIN='9(8)',PICOUT='9(8)'
          DFHMDF POS=(03,48),LENGTH=11,                           *
          INITIAL=' 처리일자 :'
OPENYMD   DFHMDF POS=(08,48),LENGTH=08,ATTRB=(NUM,FSET),          *
          PICIN='9(8)',PICOUT='9(8)',INITIAL='00000000'
CHRYMD    DFHMDF POS=(03,61),LENGTH=10
          DFHMDF POS=(04,04),LENGTH=14,ATTRB=(BRT,PROT),          *
          INITIAL='CRT-ID  :'
CRTID     DFHMDF POS=(04,20),LENGTH=04,ATTRB=(IC,PROT),INITIAL='****'
          DFHMDF POS=(04,25),LENGTH=01,ATTRB=ASKIP
          DFHMDF POS=(04,34),LENGTH=12,INITIAL='CRT-MODEL :'
                              ⋮
```

그림 1. Map 파일

제안된 변수 분류는 Map 파일에 정의된 필드들의 명을 이용하여 일차적인 변수의 용도를 분석하고 DATA DIVISION 을 분석하여 변수의 구조적인 정보를 알아낸다. 그리고 로직을 구현하는 PROCEDURE DIVISION 을 통하여 연관성을 가지는 변수들을 그룹화하게 된다. 변수 유형은 입력 변수, 출력변수, 외부 참조 변수와 임시 변수로 나누게 된다. 입력 변수는 비즈니스 로직의 입력 데이터를 전달하거나 프로그램의 흐름을 결정하는 제어변수로 사용될 수 있다. 출력 변수는 실행 결과값을 보여주는 출력 이벤트와 관련이 있는 변수이다. 그들을 비즈니스 로직의 유형에 따라 존재할 수도 있고 아닐 가능성도 있다. 그림 1 에서 출력 변수는 필드의 속성이 PROT 로 설정되어 있는 변수들은 우선적으로 선택이 된다. 그리고 입력 변수는 프로그램내의 분석을 통해 얻어진다. 외부 참조 변수는 프로그램간의 호출관계가 있을 경우 데이터를 전달하는 공간에 정의된 변수들이다. 마지막 임시 변수는 일시적으로 데이터 저장하는 역할을 하는 변수이다.

2.2 워크플로워 분석

본 논문에서 사용되는 컴포넌트 래핑의 단위는 재사용 가능한 모듈을 포함하고 있는 하나의 워크플로워이다. 이것을 식별하기 위해서는 시스템과 프로그램 수준의 정보를 모두 요구하게 된다. 여기에서는 프로그램 간의 흐름(Inter-program flow)과 프로그램 내부 흐름(Intra-program control flow)으로 나뉘어진다.

2.2.1 Inter-program control flow

프로그램 간의 제어흐름(Intra-program control flow)은 시스템 수준에서 프로그램간의 흐름을 식별하는 것이다. CICS 용 COBOL 시스템은 JCL(Job Control Language) 파일을 분석하거나 소스 파일을 분석하여 프로그램간의 호출관계를 알아낼 수 있다. 본 논문에서는 소스파일을 이용하여 CALL 문, EXEC LINK 문과 EXEC XCTL 문의 정보를 이용하여 얻어낸다.

2.2.2 Intra-program control flow

프로그램 내부의 제어흐름은 프로그램 수준에서 파라그래프 간의 흐름 정보를 말한다. 프로그램의 파라그래프간의 흐름은 순차적인 흐름과 분기 흐름 두 가지로 나눌 수 있다. 순차적인 흐름은 분기 명령에 관계없이 코드에 기술한 순서대로 실행되는 일련의 흐름을 말하고 분기 흐름은 분기 명령문에 의해 발생되는 흐름을 말한다. 그리고 분기 명령문에는 PERFORM 문과 같은 구조적인 제어 관계, GOTO 문과 같은 비구조적인 제어 관계가 있을 수 있다. 본 논문에서는 프로그램내의 파라그래프 흐름을 찾아내기 위해 두개의 흐름을 조합하여 분석하게 된다. 그리고 그 분석을 통하여 프로그램 내의 파라그래프간의 호출 관계과 실행 경로를 트리 형태로 생성해 주는 그림 2 와 같은 호출 트리를 생성한다.

호출 트리는 트리 구조를 이용하여 IF 문이나 호출문에 의해 발생되는 모든 경우의 흐름을 표시해 준다. 그림 2 에서 각 노드의 정규화 표현식은 다음과 같다.

<파라그래프명>[{제어변수}+ {[%] 피호출파라그래프명}+]

여기서 '%' 는 GOTO 에 의해 호출 되는 경우에 표시된다. 따라서, 그림 2 의 상단 부분을 보면 2000-RECV-SECTION 이라는 파라그래프는 0000-SECTION-CONTROL 파라그래프에서 CA-SECT-CODE 라는 제어 변수에 의해 GOTO 형태로 실행되는 경로를 보여준다. 프로그램 내의 로직 흐름들은 호출 트리를 이용하여 분석하게 된다.

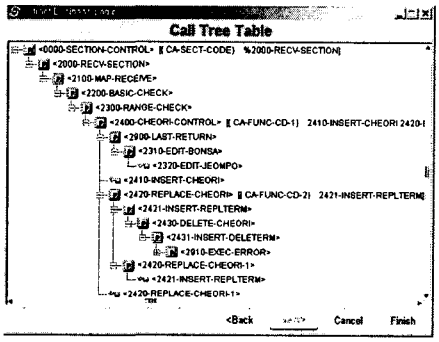

그림 2. 호출 트리

2.3 슬라이싱 판별 기준

프로그램의 슬라이싱은 프로그램 조각(Slice)를 계산하는 것을 말한다. 그리고 프로그램 조각은 슬라이싱 판별 기준을 이용하여 서로간의 영향을 주고 받는 부분들로 구성이 된다. 일반적으로 슬라이싱 판별 기준은 프로그램의 라인 번호와 변수들로 구성이 된다[1]. 프로그램 슬라이싱은 프로그램을 이해하거나 디버깅할 때 유용하게 사용되는 기술이다. 하지만 이 기술은 프로그램 내의 추상화된 정보를 포함하고 있는 비즈니스 로직을 식별하기에는 충분하지 않다. 따라서 사용자의 개입이 반드시 필요하게 된다. 본 논문에서 제안한 슬라이싱 판별 기준은 비즈니스 로직의 적합도를 측정하는 것이다. 측정하고자 하는 단위는 워크플로워 단위가 된다. 비즈니스 로직의 적합도(M)는 사전에 사용자로부터 입력 받은 비즈니스 로직의 유형들과 측정하고자 하는 워크 플로워간의 유사도를 표현한 것이다.

$$M(S,V)=\sum_{i\in S}\sum_{j\in V}W(S_i,V_j)$$

S 는 하나의 워크 플로워에 존재하는 명령문들의 집합을 나타낸다. 변수 분류에 의해 얻어진 변수들의 집합 V 은 입력, 출력, 외부 참조 집합 군이 된다.
W 는 하나의 명령문 에서 변수 집합 군과 그 변수들의 Use/Define 관계를 측정하기 위한 가중치 함수이다. 가중치 함수는 크게 변수에 대한 조작과 데이터베이스에 대한 조작으로 나눌 수 있다. 전자는 변수간의 할당(Assignment), 비교(Comparison), 연산(Operation)에 관련을 가진다. 후자는 데이터베이스의 읽기(Read), 쓰기(Write), 삭제(Delete), 갱신(Update)와 관련을 가진다.

3. 컴포넌트 래핑

식별된 워크 플로워는 한 개의 EJB 컴포넌트로 생성된다. 하지만 EJB 기반의 컴포넌트 프레임워크와 COBOL 시스템의 프레임웍이 상이하므로 바로 연계될 수 없다. 제안된 레거시 래퍼 아키텍처는 중간 프레임웍을 이용하여 상이한 두 프레임워크를 쉽게 연계할 수 있도록 도와준다. 래퍼 아키텍처는 그림 3 와 같이 계층적 구조로 이루어 진다.

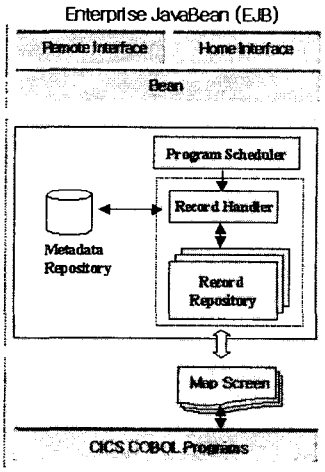

그림 3. 레거시 래퍼 구조

래퍼 아키텍처는 EJB 프레임웍, 중간 프레임웍, 레거시 프레임웍으로 이루어져 있다. 중간 계층의 프레임웍은 다음 네 부분들로 구성된다.
레코드 저장소는 Map 화면의 정보를 객체로 저장하고 있으며 EBJ 시스템이 CICS 시스템으로 데이터를 전달하기 위한 일시적인 저장소 역할을 담당한다.
메타 저장소는 COBOL 의 필드들이나 레코드들에 대한 메타 정보를 가지고 있다. 메타정보는 그림 1 에서 정의한 Map 파일 정보로 구성되며 Map 집합명과 Map 명의 조합으로 고유한 키를 가지게 된다. 그리고 메타 저장소는 런타임으로 메타정보를 제공한다. 레코드 핸들러는 EJB 로부터의 명령을 해석하여 메타 저장소로부터 런타임으로 메타정보를 얻는다. 그리고 레코드 저장소로부터 원하는 정보를 추출해낸다. 프로그램 스케쥴러는 식별된 워크 플로워에 포함되어 있는 연결성을 가지고 있는 프로그램들의 항해정보를 가지고 있다.

4. 레거시 컴포넌트 검증

소프트웨어의 기능을 테스팅하는 주요 프로세스는 테스트케이스와 데이터 생성, 테스팅하고 결과를 비교하는 것이다. 그리고 테스트 케이스 와 데이터 생성은 소프트웨어 테스팅 과정에서 많은 시간을 소요하게 된다. 본 논문에서는 래퍼 컴포넌트를 검증하기 위해 Black box 테스팅을 하게 된다. 그리고 테스트 케이스는 EJB 의 홈 인터페이스와 원격(Remote) 인터페이스를 테스팅하게 된다. 전체 테스팅 환경은 그림 4 와 같다. 테스트 케이스와 테스트 데이터를 XML 문서로 테스트 서버로 전달하게 되면 테스트 서버는 인터페이스 테스팅과 성능 테스팅을 하고 테스트 결과를 리포팅하게 된다.

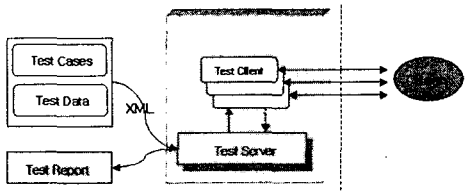

그림 4. 테스팅 환경

4.1 테스트 데이터 생성

테스트 데이터는 사용자가 수작업으로 할 수 있지만 데이터 양이 많을 경우에는 불가능하게 된다. 따라서 무작위(Random)로 데이터 입력을 자동화 할 수 있어야 한다. 그리고 데이터에는 일정한 제약조건(Constraint)를 가지고 있을 수 있기 때문에 이런 조건을 고려해야 한다. 데이터의 제약 조건은 그림 1 에서 Map 파일을 통해 정보를 얻을 수 있다. 만약 KEYGRP2 일 경우는 "9(8)"로 표현되었다면 이는 8 자리 정수임을 알 수 있다. 따라서 Map 파일의 화면 디스플레이 정보를 분석하여 테스트 데이터를 자동 생성한다.

4.2 XML 파일 생성

본 논문의 테스트 서버는 컴포넌트의 동적 테스팅을 지원하기 위하여 테스트 케이스와 테이트를 XML 문서로 입력 받아 자동으로 테스트 클라이언트 객체를 이용하여 테스팅을 하게 된다.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <testcases>
  + <class name="Converter">
  - <class name="Converter">
    - <method name="yenToEuro">
        <param name="arg1" type="java.math.BigDecimal">1000</param>
        <return type="java.math.BigDecimal" />
      </method>
    </class>
  - <class name="ConverterHome">
    - <method name="create">
        <param name="arg1" type="void" />
      - <return type="Converter">
          <class name="Converter" />
        </return>
      </method>
    </class>
</testcases>
```

그림 5. 테스팅을 위한 XML 파일

5. 결론

레거시 시스템 자원을 재활용하는데 있어서 기존 시스템을 이해하고 재사용하기 위한 모듈을 마이닝하는 작업이 필수적이다. 본 논문에서 COBOL 로 구현된 시스템에서 재활용성이 높은 비즈니스 로직을 식별하고 재사용할 수 있는 기법을 제시하고 변환된 시스템의 검증하는 시스템을 구현하였다. 추후로 XML 을 이용한 테스팅을 자바의 표준화 XML 바인딩인 JAXB 를 이용하여 보다 자동화되고 유연한 테스팅 환경 제공이 요구된다.

참고문헌

[1] K. Kawabe, Variable Classification Technique for Software Maintenance and Application to The Year 2000 Problem, *Proc. 6th Software Engineering Conf. (ASPEC'99)*, 1999, 500-506.

[2] K. Kawabe, Variable classification technique for software maintenance and application to the year 2000 problem, *Proc. 2nd Software Maintenance and Reengineering*, 1998, 11-19.

[3] J. K. Joiner, Data-Centered Program Understanding, *Proc. Software Maintenance*, 1994, 272-281.

[4] X. P. Chen, Automatic variable classification for COBOL programs, *Proc. 18th, Computer Software and Application Conf.(COMPSAC 94')*, 1994, 432-437.

[5] J. Q. Ning, Recovering reusable components from legacy systems by program segmentation, *Proc. Reverse Engineering*, 1993, 64-72.

[6] H. M. Sneed, Extracting business logic from existing COBOL programs as a basis for redevelopment, *Proc. 9th, Program Comprehension(IWPC 2001)*, 2001, 167-175.

[7] H. M. Sneed, Extracting business rules from source code, *Proc. 4th, Program Comprehension*, 1996, 240-247.

[8] H. Huang, Business rule extraction from legacy code, *Proc. 20th, Computer Software and Applications Conf.*, 1966, 922-926.

[9] A. Perkins, Business rules=meta-data, *Proc. 34th, Technology of Object-Oriented Languages(TOOLS 34')*, 2000, 285-294.

[10] D. C. C. Poo, Events in use cases as a basis for identifying and specifying classes and business rules, *Proc. Technology of Object-Oriented Languages*, 1999, 204-213.

[11] P. Lang, Modeling business rules with situation/activation diagrams, *Proc. 13th, Data Engineering*, 1997, 455-464.

[12] H. M. Sneed, A Case Study in Software Wrapping, *Proc. IEEE Software Maintenance*, pp86-92, 1998

[13] M. S. Lee, The design and implementation of Enterprise JavaBean (EJB) wrapper for legacy system, *Systems, Man, and Cybernetics, 2001 IEEE International Conf*, 2001, 1988 - 1992 vol.3