레거시 시스템으로부터 비즈니스 로직 식별

이 문 수, 양 영 종
한국전자통신연구원(ETRI)
(mslee, yjyang)@etri.re.kr

Business Logic Identification in Legacy System

Moon-Soo Lee[0] Young-Jong Yang

Dept. of ETRI-Computer & Software Technology Lab.

요 약

레거시 시스템은 수년간 기업에서 많은 노력과 투자하여 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.

본 논문에서는 레거시 시스템을 컴포넌트 래핑 기술을 이용하여 엔터프라이즈 자바 빈(EJB)으로 생성하는 지원도구 개발의 일환인 레거시 컴포넌트 식별 기법을 소개한다. 제안된 식별 기법은 비즈니스 로직을 변수 분류(Variable Classification), 슬라이싱 판별 기준, 워크플로워 분석을 이용한 레거시 컴포넌트 후보를 식별하는 방법을 제시한다.

1. 서 론

레거시 시스템은 기업 전략에 중요한 요소이며 이들은 COBOL이나 PL/I과 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 새로운 컴포넌트로 패키징해 줄 수 있는 자동화 된 재공학 도구가 요구된다.

기존 시스템으로부터 재사용 할 수 있는 기능을 식별하는 것은 기업 업무에서 중요한 비즈니스 로직(Business Logic)을 찾는 것과 유사하다. 비즈니스 로직은 기업의 비즈니스 정책을 포함하고 있고 여러 해 동안 새로운 비즈니스 환경에 맞게 적응해 나가며 유지되어 왔다. 하지만 시스템의 프로그램 소스 코드에서 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. 프로그램 분해 기법은 소스 코드에서 관련성이 있는 부분들을 식별하기 위해서 반드시 사용자의 개입이 요구되어 진다. Hung[8]은 영역 변수(Domain Variable) 식별 단계, 슬라이싱 판별 기준 식별 단계, 일반화된 프로그램 슬라이싱 단계등 3단계들로 나누었다. 그리고 각 단계마다 heuristic한 기법들을 제시하였다. Sneed[7]은 대부분의 비즈니스 로직은 출력 값들과 밀접한 관련이 있다는 가정한다. 이러한 가정을 통하여 출력 값과 프로그램 분할(Stripping)을 이용하여 식별하는 기법을 제안하였다. 일반적으로 소스코드로부터 비즈니스 로직을 식별 할 때 데이터와 그들의 흐름을 이용한다.

본 논문에서는 비즈니스 로직과 관련성을 가지고 있는 데이터를 찾기 위해서 변수 분류(Variable Classification)법을 통하여 비즈니스 로직을 대표하는 영역 변수들에 대한 후보들을 식별해낸다. 변수 분류법으로부터 얻어진 정보를 통하여 사용자는 선택된 영역 변수들로 찾고자 하는 비즈니스 로직의 유형들을 다양한 비즈니스 로직 구성요소의 가중치 설정을 통하여 슬라이싱 판별 기준을 정의하게 된다. 마지막 단계에서는 식별된 비즈니스 로직을 컴포넌트 래핑을 하기 위해서 워크플로워를 식별하게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 재사용 가능한 모듈 식별에 대한 내용을 제시하였다. 3장에서는 식별된 비즈니스 로직을 컴포넌트 래핑으로 캡슐화하기 위한 워크플로워 분석에 대한 내용을 다루게 된다. 마지막 4장은 결론과 추후과제에 대해 설명할 것이다.

2. 재사용 기능 식별

재사용 모듈을 식별은 크게 2단계로 구성되어 있다. 첫번째 단계는 변수 분류 단계로 우선 비즈니스 로직과 관련성을 가지는 변수들을 그룹화한다. 각 변수 그룹을 입력 변수, 출력변수, 외부 참조 변수들로 분류한다. 다음 단계에서는 사용자가 재사용하고자 하는 비즈니스 로직의 유형을 슬라이싱 판별 기준으로 도구에 설정하게 된다. 설정된 판별 기준은 프로그램의 최소 모듈, 파라그래프 단위로 측정이 이루어진다. 그리고 파라그래프 내에서 각 명령문(Statement)들의 적합도를 계산하고 사용자에게 제시해주게 된다.

2.1 변수 분류(Variable Classification)

변수 분류는 사전에 정의된 분류 그룹에 따라 프로그램에서 사용된 변수들을 그룹화하는 것이다. 이 단계는 변수의 가시성(visibility)을 가지고 있지 않는 COBOL프로그램에 대해 사용자가 유지보수 하거나 이해하는데 많은 도움을 주게 된다. Kawabe[1,2]는 Y2K문제를 해결하기 위해 COBOL의 프로시저와 구조적인 분석을 통하여 변수 유형을 그룹화 하였다. Joiner[3]은 영역변수, 입력 변수, 프로그램 변수, 국지적 변수, 전역변수, 출력변수, 제약조건 변수, 제어변수와 같이 8개의 변수 형태를 정의하고 그룹화하였다.
일반적으로 비즈니스 로직은 임의의 입력 변수 집합(x)과 출력 변수 집합(y)으로 표현되는 하나의 방정식으로 표현할 수 있다.

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

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.
IBM CICS시스템에서 구현된 COBOL프로그램의 입출력은 BMS(Basic Mapping Support) MAP파일에 정의한다. BMS는 물리적으로 다양한 터미널에 상관하지 않고 개발자는 MAP 프로그램에 전념할 수 있도록 도와준다. 그림 1은 CICS COBOL 프로그램의 전처리(Preprocessor) 과정을 보여준다.

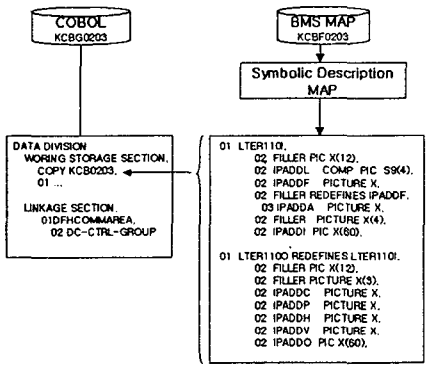

그림 1. COBOL Preprocessor

본 논문은 우선 COBOL프로그램내의 데이터 선언부를 분석하여 변수의 구조적인 정보를 알아낸다. 그리고 로직을 구현하는 프로시져 선언부를 통하여 연관성을 가지는 변수들을 분류하게 된다.

2.1.1 입력 변수

입력 변수는 프로그램에서 키보드 입력과 같은 이벤트와 관련성을 가지는 변수이다. 그들은 비즈니스 로직의 입력 데이터를 전달하거나 프로그램의 흐름을 결정하는 제어변수로 사용될 수 있다. 입력 변수는 Symbolic description map파일명의 마지막 문자가 ' I' 로 명명된 필드 명으로 식별할 수 있으며 데이터 흐름 분석을 통하여 관련된 변수들을 그룹화한다.

2.1.2 출력 변수

출력 변수는 실행 결과값을 보여주는 출력 이벤트와 관련이 있는 변수이다. 그들을 비즈니스 로직의 유형에 따라 존재할 수도 있고 아닐 가능성도 있다. 출력 변수는 Symbolic description map파일명의 마지막 문자가 ' O' 로 명명된 필드 명으로 식별할 수 있으며 데이트 흐름 분석을 통하여 관련된 변수들을 그룹화한다.

2.1.3 외부참조 변수

하나의 프로그램이 다른 프로그램을 호출할 경우 프로그램간의 데이터 전송이 필요하다. COBOL프로그램에서는 데이터 선언부의 Linkage Section부분에 데이터를 정의하게 된다. 이 영역에 선언된 변수 및 관련성을 가지는 변수들을 그룹화한다.

2.2 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 최소 단위는 파라그래프 단위가 된다. 비즈니스 로직의 적합도(M)는 사전에 사용자로부터 입력 받은 비즈니스 로직의 유형들과 측정하고자 하는 파라그래프간의 유사도를 표현한 것이다.

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

S 는 하나의 파라그래프에 존재하는 명령문들의 집합을 나타낸다. 변수 분류에 의해 얻어진 변수들의 집합 V은 입력, 출력, 외부 참조 집합 군이 된다. W는 하나의 명령문 S_i에서 변수 집합 군과 그 변수들의 Use/Define관계를 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. 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.

3.1 Inter-program control flow

프로그램 간의 제어흐름(Intra-program control flow)은 시스템 수준에서 프로그램간의 흐름을 식별하는 것이다. CICS용 COBOL시스템은 JCL(Job Control Language) 파일을 분석하거나 소스 파일을 분석하여 프로그램간의 호출관계를 알아낼 수 있다. 본 논문에서는 소스파일을 이용하여 CALL문, EXEC LINK문과 EXEC XCTL문의 정보를 이용하여 얻어낸다.

3.2 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 구조적인 제어 관계와 비구조적인 제어 관계가 있을 수 있다. 본 논문에서는 프로그램내의 파라그래프 흐름을 찾아내기 위해 두개의 흐름을 조합하여 분석하게 된다.

3.2.1 구조적 호출 관계

구조적 호출 관계는 PERFORM문에 의한 호출 관계를 의미한다. 이 명령문으로 호출된 파라그래프는 모듈화 되어져 있다. Call Graph는 파라그래프 간의 호출관계는 계층적인 구조를 시각적으로 보여 준다. 그림 2에서 최상위 노드는 프로그램명이 되고 PERFORM관계는 직선으로 표시된다.

3.2.2 비구조적 호출 관계

비구조적 호출 관계는 GOTO문에 의한 파라그래프의 흐름을 말한다. 이 관계도 Call Graph에서 계층적인 구조로 표현될 수 있으며 그림 2에서 굵은 선으로 표시된다

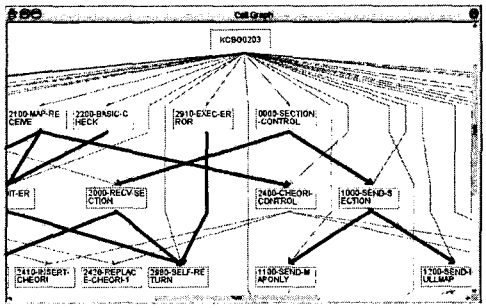

그림 2. Call Graph

3.2.3 호출 트리

전 단계의 구조적 및 비구조적 호출관계를 이용하여 프로그램 수준의 실행 경로를 알아낼 수 있고 이를 그림 3과 같이 호출 트리(Call Tree)로 표현 할 수 있다. 호출 트리는 트리 구조를 이용하여 IF문이나 호출문에 의해 발생되는 모든 경우의 흐름을 표시해 준다. 그리고 트리의 최하위 노드는 앞 단계의 슬라이싱 판별 기준을 통해 얻어진 파라그래프가 된다.

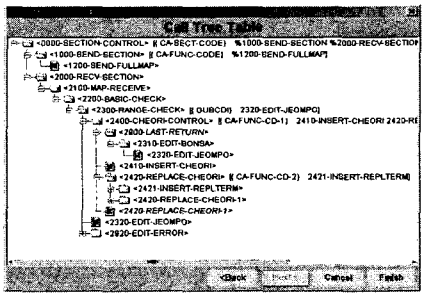

그림 3. Program Call Tree

3.3 워크플로워 식별

선택되어진 파라그래프는 사용자가 원하는 비즈니스 로직을 포함하고 있으며 이러한 파라그래프를 실행하기 위한 워크플로워 식별이 필요하다. 우선 워크플로워의 식별은 프로그램내의 흐름을 찾아내고 다음 프로그램간의 흐름을 알아내게 된다.

3.3.1 프로그램내의 워크 플로워

프로그램내의 워크플로워는 호출 트리 그래프를 이용하여 비즈니스 로직을 대표할 수 있는 파라그래프를 실행할 수 있는 경로를 찾게 된다. 그리고 그 경로를 실행하기 위한 변수, 즉 제어변수의 리스트를 찾아낸다. 제어변수는 주로 입력에 의해 좌우되며 이는 사용자의 입력 변수일 수도 있고 호출 프로그램에 의해 전달된 변수일 수도 있다.

3.3.2 프로그램간의 워크 플로워

만약 선택된 파라그래프가 속해 있는 프로그램이 피호출 프로그램일 경우에 호출 프로그램에 대한 워크플로워를 분석해야 한다. 호출 프로그램의 워크플로워는 호출 명령문과 피호출 프로그램의 제어변수를 이용하여 영향 분석을 통하여 전체 워크플로워를 찾아내게 된다.

4. 결론

레거시 시스템 자원을 재활용하는데 있어서 기존 시스템을 이해하고 재사용하기 위한 모듈을 마이닝하는 작업이 필수적이다. 본 논문에서 COBOL로 구현된 시스템에서 재활용성이 높은 비즈니스 로직을 식별하고 재사용할 수 있는 기법을 제시하였다. 현재 연구는 기존 시스템의 변화를 주지 않고 재사용할 수 있는 방법을 제시하였으며 추후 변환된 시스템의 성능에 관한 연구도 요구된다.

5. 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