C 언어로부터 C++ 객체 생성과 주기의 결정에 관한 연구

최정란, 이문근
전북대학교 컴퓨터 과학과
e-mail : {jlchai, mklee}@cs.chonbuk.ac.kr

A Study on Determination of Instantiations and Life Cycles of C++ objects from C code

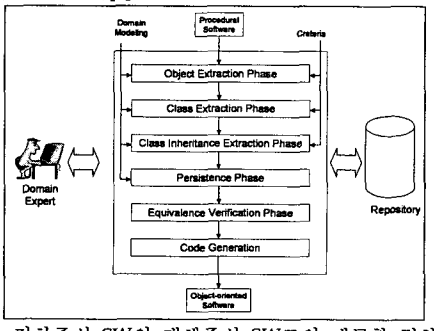
Joeng-ran Choi, Moon-kun Lee
Dept. of Computer Science, Chonbuk National University

요 약

절차지향 소프트웨어를 객체지향 소프트웨어로의 재공학은 소프트웨어 유지·보수 비용의 절감과 기존의 시스템에 새로운 요구사항을 수용하는 등 많은 장점이 있다. 본 논문은 절차지향 소프트웨어를 재공학하는 과정에서 안전한 지속성(*Safe Persistency*) 결정 문제, 즉 객체의 생성과 생성 주기의 결정 방법을 제안하였다. 본 논문에서는 지속성 결정을 위해 다섯 단계의 모델링 과정을 제시하였으며 모델링 과정을 통해 객체의 정확한 생성과 소멸 시점을 추출하였고 정제 과정을 거침으로써 객체의 메시지 전달과 생성/소멸 과정에서 안정성과 일관성을 유지할 수 있도록 하였다.

1. 서론

소프트웨어는 제작된 후 지속적인 유지·보수가 필요하다. 이러한 흐름에 맞추어 시스템을 재공학할 때 유지·보수 및 재사용 측면에서 기존의 방식들보다 유리한 객체지향 파라다임을 적용하여 기존의 시스템으로 재개발한다면, 소프트웨어 생산성을 향상시킬 수 있고, 소프트웨어의 유지·보수 비용을 절감할 수 있으며, 시스템에 새로운 요구를 수용할 수 있게 되는 등 많은 장점을 가지게 된다[1].



<그림 1> 절차중심 SW의 객체중심 SW로의 재공학 절차 흐름도

일반적으로 재공학 방법은 <그림 1>과 같은 절차로 진행되는 것이 바람직하다. 객체 추출 단계에서는 전역 변수, 사용자 자료형 함수와 파라미터를 기반으로 관련성 정도를 기준으로 클러스터링한다. 클래스 추출 단계에서는 클러스터링된 객체 후보들의 공통적인 특성을 추출하여 클래스를 추출한다. 클래스 추출 단계의 결과는 클래스들간에 대등한 평면 관계를 이루고 있다. 상속 관계 추출 단계는 전 단계에서 추출된 평면화된 클래스들의 공통적 특성을 추출하여 *part-of* 관계나 *is-a* 관계를 추출하여 계층 구조를 만든다. 지속성 단계에서는 절차 지향 프로그램에는 존재하지 않는 객체 지향 특성들과 동적인 부분을 추가한다. 이러한 것들은 클래스의 생성자, 소멸자, 동적 메모리 할당/해제 등이 있다. 동일성 검증 단계에서는 생성된 객체지향 프로그램이 정상적으로 작동하는지와 원래의 절차지향 프로그램과 변환된 객체지향 프로그램이 의미상으로 동등한지에 대한 검사를 한다. 코드 생성 단계에서는 전단계에서 생성된 뼈대 코드(skeleton code)를 기반으로 실행 가능한 객체 지향 프로그램을 생성한다.

본 논문에서는 위 재공학 과정 중 네 번째 단계인 지속성 결정 방법을 기술한다. 객체지향 프로그래밍에서 객체를 생성하는 방법은 모든 객체는 초기에 생성하여 프로그램이 종료시 소멸하는 방법과 객체가 필요시 생성되었다가 더 이상 쓰이는 곳이 없을 때 소멸되는 방법의 두 가지가 있다. 초기 객체 생성은 모든 필요한 객체가 이미 존재하고 있다는 점에서 안전성이 보장되지만 불필요한 메모리 낭비로 인하여 과부하(overload)를 초래하게 된다. 반대로, 객체들의 정확한 생성과 소멸 시점을 파악함으로써 생성되지 않은 객체간의 메시지 교환(interaction) 발생 문제를 없앨 수 있어서 전체 시스템의 안전성(safety)과 일관성(consistency) 및 정확성(correctness)을 높일 수 있다. 본 논문에서는 상속성 추출 결과 생성된 클래스들이 실제 프로그램에서 효율적으로 객체를 생성하고 소멸하는 과정을 거침으로써 시스템의 부하를 최소화 할 수 있도록 하였다.

본 논문에서 사용한 지속성 결정 방법은 다음과 같이 5가지 모델링에 기초를 두고 있다: 정적 정보(*Static Information*), 투영(*Projection*), 반영(*Reflection*), 인스턴스(*Instantiation*) 및 정제(*Refinement*).

본 논문의 구성은 다음과 같다. 2절에서는 다섯 단계의 모델링 과정을 기술하며, 3절에서는 결론 및 향후 연구에 대해 기술하였다.

2. Persistency Modeling

객체들의 지속성을 결정하기 위하여 본 논문에서는 5 단계의 모델링 과정을 제시한다. 모델링 과정을 살펴보면 다음과 같다: 1) 정적 정보 모델링 단계에서는 프로그램의 정적 정보를 추출하는 과정으로 FTV 그래프[2]에 시간의 개념을 첨가하여 확장한 그래프 형태인 FTV-time 그래프를 생성, 2)투영 모델링 단계는 정적 정보 모델링 결과 생성된 FTV-time 그래프를 상속성 추출 결과 생성된 클래스들과의 투영을 통하여 서로 관련성이 큰 함수·자료형·변수를 그룹화, 3)반영 모델링 단계는 2단계의 투영 모델링 결과 생성된 그룹을 함축하여 객체로 표현, 객체의 시작/종료 시점을 추출, 4) 인스턴스 모델링 단계는 3단계 결과에 객

본 연구는 한국 과학재단 특정기초연구(과제번호 1999-2-2-303-003-3) 지원으로 수행되었음

체의 생성과 소멸 시점을 추출, 5)정제 모델링 단계에서는 4단계의 과정을 통하여 추출된 생성/소멸 시점에 대해 안전성과 일관성을 유지하기 위해 정제하는 모델링 과정이다.

2.1 정적 정보 모델링(Static Information Modeling)

지속성을 결정하는 5단계 중 첫 번째 단계인 정적 정보 모델링은 절차지향 프로그램에서 함수(F), 자료형(T), 변수(V)들의 상호작용을 상대적 시간에 기반하여 모델링하는 과정이다. 모델링 결과 생성된 그래프인 FTV-time 그래프($G^{FTV-time}$)는 객체 추출과정에서 사용되었던 FTV 그래프(G^{FTV})를 확장한 그래프이다 [2]. G^{FTV}에서 노드로 표현되었던 함수, 변수, 자료형이 $G^{FTV-time}$에서는 이들 노드의 상호작용이 시간의 변화에 따라 여러 노드와 에지로 나타나게 되어 이들의 주기를 가시적으로 나타내고 있다. 그래프 $G^{FTV-time}$의 형태는 다음과 같이 3-쌍의 튜플로 정의된다.

$$G^{FTV-time} = \langle N^{FTV-time}, E^{FTV-time}, S^{FTV-time} \rangle$$

노드 $N^{FTV-time}$는 함수, 자료형, 변수와 관련된 노드의 집합이고, 이를 수식으로 표현하면 다음과 같다.

$$N^{FTV-time} = N_{func}^{FTV-time} \cup N_{type}^{FTV-time} \cup N_{var}^{FTV-time}$$

노드와 노드를 연결하는 에지의 집합은 함수, 자료형, 변수를 개별적으로 연결해주는 3가지의 수직 방향 에지와, 함수의 호출(call), 반환(return), 읽기(read), 쓰기(write), 및 인스턴스(instance)에 관련된 5가지의 수평 방향 에지의 집합이다. 이를 수식으로 표현하면 다음과 같다.

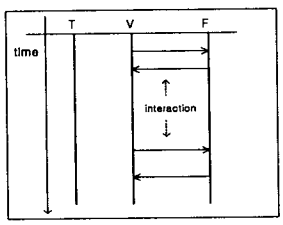
$$E^{FTV-time} = E_{persistence}^{FTV-time} \cup E_{call}^{FTV-time} \cup E_{return}^{FTV-time} \cup E_{read}^{FTV-time} \cup E_{write}^{FTV-time} \cup E_{instance}^{FTV-time}$$

단, $E_{persistence} = \{e_{persistence} \mid e_{persistence} \in (E_{func} \vee E_{type} \vee E_{var})\}$

$S^{FTV-time}$는 메시지 교환의 순서를 의미하고 발생하는 에지의 순서와 일치한다. 순서(sequence)를 표현함으로써 에지들간의 상대적 위치를 알 수 있고 시간의 흐름을 가시적으로 볼 수 있다는 장점을 가진다. $S^{FTV-time}$를 수식으로 표현하면 다음과 같다.

$$S^{FTV-time} = \langle \ldots e_i <_t e_j \ldots \rangle \text{ 단, } e_i, e_j \in E^{FTV-time}$$

여기에서 $<_t$는 상대적 시간 순서를 의미하여 e_i가 e_j보다 시간적으로 순서가 앞선다. <그림 2>는 정적정보 모델링 결과 생성된 그래프이고 함수와 변수간에 밀접한 메시지 교환이 있음을 알 수 있다.



<그림 2> FTV-time 그래프($G^{FTV-time}$)

2.2 투영 모델링(Projection Modeling)

투영 모델링은 정적 정보 모델링 단계에서 추출된 $G^{FTV-time}$를 상속성 추출결과 생성된 클래스와의 매핑 과정을 통해 관련성이 큰 함수, 자료형, 변수를 그룹화하는 과정이다. $G^{FTV-time}$를 클래스와 투영 모델링 결과 투영 FTV-time 그래프($G_{projection}^{FTV-time}$)가 추출되고 다음과 같이 3-쌍의 튜플로 정의된다.

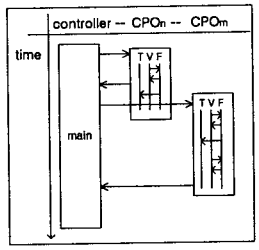
$$G_{projection}^{FTV-time} = \langle N_{projection}^{FTV-time}, E_{projection}^{FTV-time}, S_{projection}^{FTV-time} \rangle$$

$G_{projection}^{FTV-time}$는 투영 결과 여러 개의 그룹으로 묶여지며, 그룹화된 $G_{projection.i}^{FTV-time}$는 *소스코드에서 클래스를 투영한 객체(CPO: CP-Object 또는 a classed projected object in source code)*라 한다. 투영된 결과 발생하는 $N_{projection}^{FTV-time}$은 그룹화된 $G_{projection.i}^{FTV-time}$의 집합으로 표현되며 이를 수식으로 표현하면 다음과 같다.

$$N_{projection}^{FTV-time} = \{G_{projection.i}^{FTV-time} \mid G_{projection.i}^{FTV-time} \subseteq G^{FTV-time}, G_{projection.i}^{FTV-time} \cap G_{projection.j}^{FTV-time} = \emptyset, i \neq j, 1 \le i \le \#obj\}$$

단, $G_{projection.i}^{FTV-time} \subseteq G^{FTV-time}$ $(1 \le i \le n)$ n: object의 개수

에지의 집합은 $G^{FTV-time}$의 것과 같은 종류로 발생되고 $G^{FTV-time}$의 에지와 포함관계를 가지고 있다. 순서 $S_{projection}^{FTV-time}$는 $G^{FTV-time}$와 마찬가지의 순서를 가지고 있으며 $S_{projection}^{FTV-time}$가 $S^{FTV-time}$에 포함된다. <그림 3> 투영 모델링 결과 생성된 그래프이다. CPO_n과 CPO_m은 그룹화된 $G_{projection.i}^{FTV-time}$를 의미한다.



<그림 3> 투영 FTV-time 그래프($G_{projection}^{FTV-time}$)

2.3 반영 모델링(Reflection Modeling)

지속성을 결정하는 3단계는 반영 모델링(Reflection Modeling)이다. 이 단계는 CPO 내부의 메시지 교환을 하나의 직선으로 표현하고 CPO 사이의 메시지 교환에 초점을 두었다. 지속성을 결정하는 목적 중의 하나가 객체의 정확한 생성과 소멸 시기를 추출하는 것이므로 내부의 메시지 교환은 이차적인 문제이다. 반영 모델링 결과 생성된 그래프를 반영 FTV-time 그래프($G_{reflection}^{FTV-time}$)라하고 다음과 같이 3-쌍의 튜플로 정의된다.

$$G_{reflection}^{FTV-time} = \langle N_{reflection}^{FTV-time}, E_{reflection}^{FTV-time}, S_{reflection}^{FTV-time} \rangle$$

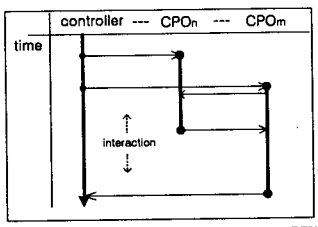
$N_{reflection}^{FTV-time}$는 CPO 사이의 메시지 교환 노드 중 CPO 사이의 최초의 메시지 교환 노드와 마지막 메시지 교환 노드의 집합을 나타내고, 수식으로 표현하면 다음과 같다.

$$N_{reflection}^{FTV-time} = \{n_{i,k} \mid 0 \le i \le \#obj, 1 \le k \le \#ip\}$$

노드와 노드를 연결해 주는 에지는 CPO 사이의 관련성이 있는 에지들의 집합이며, 이를 수식으로 표현하면 다음과 같다.

$$E_{reflection}^{FTV-time} = \{e \mid e \in (N_{reflection.i}^{FTV-time}, N_{reflection.j}^{FTV-time}), i \neq j\}$$

$S_{reflection}^{FTV-time}$는 에지의 튜플로 이루어져 있으며 이들간에는 시간적 순서가 존재한다. <그림 4>는 반영 모델링 결과 생성된 그래프이다.



<그림 4> 반영 FTV-time 그래프($G_{reflection}^{FTV-time}$)

2.4 인스턴스 모델링(Instantiation Modeling)

3단계의 반영 모델링을 통해 객체의 시작시점과 종료 시점을 알게 되었다. 그러나 객체들간의 메시지 교환이 이루어지기 전에 먼저 객체를 생성해야 한다. 그래서 4단계의 인스턴스 모델링에서는 객체의 시작시점 전과 종료 후 객체의 소멸 시점에 관련성을 추가하는 작업이 이루어지게 된다. 일련의 작업 결과 인스턴스 FTV-time 그래프($G^{FTV-time}_{instantiation}$)가 추출되고 다음과 같이 3-쌍의 튜플로 정의된다.

$$G^{FTV-time}_{instantiation} = \langle N^{FTV-time}_{instantiation},\ E^{FTV-time}_{instantiation},\ S^{FTV-time}_{instantition} \rangle$$

$G^{FTV-time}_{instantiation}$의 노드($N^{FTV-time}_{instantiation}$)는 $N^{FTV-time}_{reflection}$와 생성과 소멸에 관련한 노드를 새롭게 추가하고, 이를 수식으로 표현하면 다음과 같다.

$$N^{FTV-time}_{instantiation} = N^{FTV-time}_{reflection} \cup \{n_{i,0},\ n_{i,z}\}$$

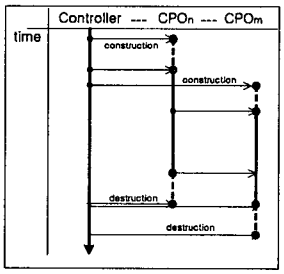
단, $n_{i,0}$, $n_{i,z}$은 생성과 소멸 노드.

에지($E^{FTV-time}_{instantiation}$)도 $E^{FTV-time}_{reflection}$와 객체들간의 생성노드를 연결한 에지($e_{constructor}$)와 소멸노드를 연결한 에지($e_{destructor}$)를 첨가한다. 이를 수식으로 표현하면 다음과 같다.

$$E^{FTV-time}_{instantiation} = E^{FTV-time}_{reflection} \cup e_{constructor} \cup e_{destructor}$$

단, $e_{constructor} \in (n_{i,0},\ n_{i,1})$ $e_{destructor} \in (n_{i,y}, n_{i,z})$.

$S^{FTV-time}_{instantiation}$는 $E^{FTV-time}_{instantiation}$의 발생 순서와 일치한다. <그림 5>는 인스턴스 모델링 결과 생성된 그래프이다.



<그림 5> 인스턴스 FTV-time 그래프($G^{FTV-time}_{instantiation}$)

2.5. 정제 모델링(Refinement Modeling)

지속성 결정의 마지막 단계는 정제 모델링이다. 앞서 네 단계를 거쳐서 생성 시점과 소멸 시점을 추출하였다. 정확한 생성과 소멸 시점을 안다는 것은 객체의 불필요한 생성과 소멸을 감소시켜 효율적으로 메모리를 사용할 수 있어 과부하를 최소화 할 수 있는 장점이 있다. 정확한 객체의 생성과 소멸을 위해 안전성과 일관성이 보장되어야 한다. 정제 모델링 단계에서는 이를 위해 제어 메소드를 추가하고 객체간의 정확한 메시지 교환을 추가하였다. 정제 모델링 결과 생성된 그래프를 정제 FTV-time 그래프($G^{FTV-time}_{refinement}$)라하고 다음과 같이 3-쌍의 튜플로 정의된다.

$$G^{FTV-time}_{refinement} = \langle N^{FTV-time}_{refinement},\ E^{FTV-time}_{refinement},\ S^{FTV-time}_{refinement} \rangle$$

그래프 $G^{FTV-time}_{refinement}$는 객체가 생성과 소멸되기 전에 먼저 요구하고 그에 대한 응답의 제어 기능을 하는 메시지 교환이 새롭게 추가된다. $N^{FTV-time}_{refinement}$은 $G^{FTV-time}_{instantiation}$의 노드와 객체간의 메시지 전달이 있는 노드, 객체 생성/소멸에 관련한 제어 노드의 집합이고, 이를 수식으로 표현하면 다음과 같다.

$$N^{FTV-time}_{refinement} = \{n_{i,k} \mid 0 \le i \le \#obj,\ 1 \le k \le \#ip\}$$

$$\cup \{n^{request}_{i,j},\ n^{ack}_{i,j} \mid j:\ number\ of\ node\}$$

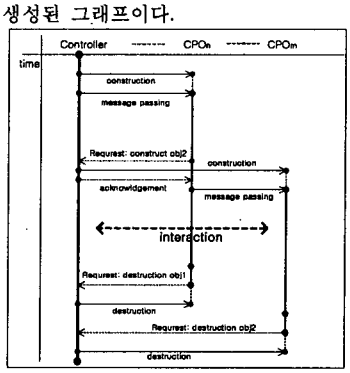
$e^{FTV-time}_{refinement}$은 $G^{FTV-time}_{instantiation}$의 에지와 객체간의 메시지 교환이 존재하는 에지(e_{object_inst})를 첨가하고, 제어 에지($e_{control}$)를 첨가하게 되며, 이를 수식으로 표현하면 다음과 같다.

$$E^{FTV-time}_{refinement} = E^{FTV-time}_{instantiation} \cup e_{object\_inst} \cup e_{control}$$

단, $e_{object\_inst} \in (n_{i,j},\ n_{k,l})$,

$e_{control} \in \{(n^{request}_{i,j}, n^{request}_{k,l}),\ (n^{ack}_{i,j}, n^{ack}_{k,l})\}$.

순서는 $E^{FTV-time}_{refinement}$의 발생 순서와 일치한다. <그림 6>은 정제 모델링 결과 생성된 그래프이다.



<그림 6> 정제 FTV-time 그래프($G^{FTV-time}_{refinement}$)

3. 결론 및 향후 연구

재공학 과정을 통하여 추출된 클래스들이 정확한 시기에 객체로 생성되고 소멸되는 과정은 객체지향 프로그램에서는 매우 중요하다. 만약 객체가 프로그램 초기부터 종료시까지 존재한다면 불필요한 메모리 낭비와 과부하를 초래하게 된다. 본 논문에서는 객체의 정확한 생성과 소멸 시점을 제시하였고 정제과정을 통해 정확한 메시지 교환을 추출하였다.

본 논문에서는 재공학 과정 중 안전한 지속성 결정에 대하여 5단계의 모델링 과정으로 나누어 기술하였다. 정적 정보에 시간 개념을 첨가하고 클래스와의 투영과정을 통해 객체로 그룹화하였으며, 반영 모델링에서는 객체의 시작과 종료시점을 추출하였다. 이를 바탕으로 인스턴스 모델링에서는 객체의 생성/소멸 시점을 추출하였고, 마지막으로 정제 과정을 통하여 객체 생성의 안정성과 일관성을 보장하였다.

향후 연구로는 생성된 객체지향 프로그램이 정상적으로 작동하는지와 원래의 절차지향 프로그램과 변환된 객체지향 프로그램이 의미상으로 동등한지를 대한 동일성 검사를 한다.

[참고문헌]

[1]Robert S. Arnold, "*Software Reengineering*," IEEE Computer Society Press, 1994.

[2]박성옥, 노경주, 이문근, "*최적합 객체 선정을 위한 다중 객체군 추출*," 한국 정보 과학회 논문집(B), 제26권, 제12호, pp. 1468-1481, 1999.

[3]박성옥, 최정란, 이문근, "*절차지향 SW를 객체지향 SW로 재공학하기 위한 클래스와 상속성 추출에 관한 연구*", 2000 한국 소프트웨어공학 학술대회 논문집, pp.51-60, 2000.02.