재사용성 향상을 위한 도메인 명세화 컴포넌트(DSC)의 설계

권영희, 조은경, 이권일
대덕대학 인터넷정보기술계열
전화 : 042-866-0275 / 핸드폰 : 011-9606-2225

Domain Specification Component Design to Improve Software Reusability

Young-Hee Kwon, Eun-Kyoung Cho, Kwon-Il Lee
E-mail : yhkwon@mail.ddc.ac.kr

Abstract

The informal specification technique lacks abstraction and preciseness. On the other hand, the formal specification technique makes the developer difficult to express and understand the software specification, because it contains mathematical expression. This paper proposes DSC(Domain Specification Component) to solves these problems. DSC supports the understanding of problem domain and improves reusability with selecting the strong point of informal and formal specification technique.

We applied the proposed DSC to CMIP-based network management manager software. And we analyzed the effects of the reusability and confirmed the increase of the reusability.

I. 서론

소프트웨어 재사용은 새로운 시스템 개발에 기존에 개발되어 있는 소프트웨어 시스템 컴포넌트들을 다시 사용함으로써 설계와 코딩 단계에서 개발기간을 단축시키게 되므로 개발비용 절감과 생산성 향상의 효과를 얻을 수 있다[1]. 현재까지는 주로 소스 코드 수준의 재사용이 연구되었으나 이는 소프트웨어 시스템의 특성과 개발 기법에 따라 코드들이 다르므로 재사용 과정 동안에 기존의 코드가 수정되어야 한다. 이 문제를 해결하기 위한 방법은 코딩 이전 단계인 분석과 설계 정보를 재사용하는 것이다[2]. 이 논문에서는 비정형적인 명세화 기법과 정형적인 명세화 기법의 한계성을 극복하고 문제 영역의 관점을 지원하여 한 도메인에 속한 다양한 어플리케이션 개발에 공통적으로 필요한 컴포넌트를 생성하여 재사용성을 향상시킬 수 있는 도메인 명세화 컴포넌트(Domain Specification Component : DSC)를 제안한다. 또한, 이를 검증하기 위해 CMIP(Common Management Information Protocol) 기반의 망 관리 매니저를 DSC에 적용하고 재사용성을 평가한다.

II. 소프트웨어 재사용과 명세화

2.1 소프트웨어의 재사용

재사용되는 소프트웨어 컴포넌트의 특성에 따라 재사용 유형을 분류하면 합성방법(compositional reuse)과 생성방법(generative reuse)이 있다[3]. 합성방법은 재사용되는 소프트웨어 컴포넌트들을 라이브러리에 저장시켜 놓은 후, 이미 만들어진 컴포넌트들을 블록으로 만들어 끼워 맞추면서 새로운 소프트웨어를 생성하는 기술이다. 생성방법은 각 응용 분야별로 소프트웨어의 공통적인 형태를 갖는 모형을 만든 후, 소프트웨어를 생성하

는 시점에서 그 모형에 적절한 매개 변수를 부여하여 알맞은 소프트웨어를 생성해 내는 방법이다.

2.2. 도메인 모델링(Domain Modeling)

비슷한 시스템들간에 공통적인 객체나 오퍼레이션을 찾아냄으로써 소프트웨어 재사용성을 향상시킬 수 있는데, 공통적인 객체나 오퍼레이션을 지니는 시스템들을 도메인이라 한다. 관련된 시스템 집합내의 객체와 오퍼레이션을 식별하고 공통성 분석을 통해 현존하는 시스템과 향후 개발될 시스템으로부터 공통적인 객체를 발견하는 과정을 도메인 분석이라 한다[4]. 도메인 모델링은 재사용 가능한 컴포넌트 라이브러리를 제공해 주고, 이런 컴포넌트들의 재사용을 통한 소프트웨어 개발은 품질, 생산성, 전체 시스템 비용 면에서 많은 이익을 가져올 수 있다.

2.3 소프트웨어 명세화

비정형적 명세는 읽고 이해하기가 쉬운 반면, 프로그램 검증이 불완전하고, 추론 단계에서 사람의 직관력 및 독창성에 의존하기 때문에 모호한 단점이 있다. 정형적 명세는 수학적인 표현을 적용하여 기술하므로 애매모호성이 없다는 이점을 갖고 있으나 사용자가 읽기 어렵다는 단점을 갖고 있다[5]. 이 논문은 객체지향 소프트웨어의 명세화 방법에 관한 연구로 컴포넌트의 인터페이스는 정형화된 방법으로 명세화하고, 시맨틱스는 비정형적 방법으로 명세화하는 텍스트 설계기법을 기반으로 하는 준 정형화 명세방법을 적용한다.

Ⅲ. 도메인 명세화 컴포넌트(DSC) 설계

3.1 DSC의 정의

컴포넌트를 정형적으로 명세화하는 가장 중요한 이유는 컴포넌트 명세를 최대한 정확하게 기술하여 구현상의 모호함을 제거하는데 있다. 따라서 제안한 DSC는 다음과 같은 요건을 충족할 수 있도록 설계되어야 한다. 첫째, 컴포넌트는 모듈화된 단위로 시스템 구조의 이해가 용이하고 독립적으로 개발가능해야 한다. 둘째, 재사용성을 높이기 위한 수단으로 템플리트를 이용하여 정의하고 세부적인 구현 사항들은 내부에 숨겨야 한다. 셋째, OMG에서 표준화하고 있는 객체 참조모델과 객체지향의 특성을 기반으로 추상화, 상속화, 일반화, 은닉화 개념을 반영하여야 한다.

3.2 DSC의 구성

객체지향 소프트웨어의 특성인 추상화, 상속화, 은닉화 등을 효율적으로 반영할 수 있도록 컴포넌트의 특성을 parameters, exports, imports, functions, equations로 나누어 그림 1과 같이 DSC를 구성한다. 표 1에서는 DSC를 구성하는 각 요소들의 기능을 기술한다.

```
<Template>::= <Class>+
<Class>::=Class<Class-identifier>
begin
   [<parameters>]
   [<exports>]
   [<imports>]
   [<functions>]
   [<equations>]
end <Class-identifier>
```

그림 1. DSC의 구조

표 1. DSC 구성 요소의 기능

구성 요소	기 능
parameters	파라미터 명
exports	다른 클래스로 상속되는 sort와 function
imports	다른 클래스에서 상속받는 sort와 function
functions	현재 클래스에서만 사용할 수 있는 은닉함수
equations	클래스의 기능

parameters에서는 현재 클래스가 다른 클래스에서 일반적으로 사용될 수 있도록 파라미터 명을 기술한다. exports와 imports에서는 객체의 가시성 리스트를 설정하여 상속성 개념을 반영할 수 있도록 하고, functions에서는 은닉화 개념을 반영할 수 있도록 한다. equations는 클래스의 기능을 설명한다. 또한, 객체의 데이터와 기능을 sorts와 functions로 추상화하여 기술한다.

Ⅳ. 적용 및 평가

이 장에서는 서로 다른 망을 기본으로 하는 망 관리를 수행하고자 할 때 이미 개발된 망 관리 소프트웨어를 재사용하여 개발 기간, 비용 및 안정화에 도움이 될 수 있도록 하기 위해 CMIP 기반의 망 관리 매니저 관리 객체(MO : Managed Object)들을 DSC에 적용하여 명세화하고 재사용성을 측정한다.

4.1 망 관리 매니저의 재사용

CMIP 기반의 망 관리 매니저는 종속망 내의 망 요소들에 대하여 신호 및 장치에 대한 장애를 검출하고, 망 요소의 구성변경을 제어 관리하며, 성능에 대한 지속적 감시, 제어, 관리하여 망 관리에 적절한 조치를 취할 수 있는 OS의 역할을 수행한다. 이러한 기능들을 소프트웨어 구조로 표현하면 그림 2와 같다.

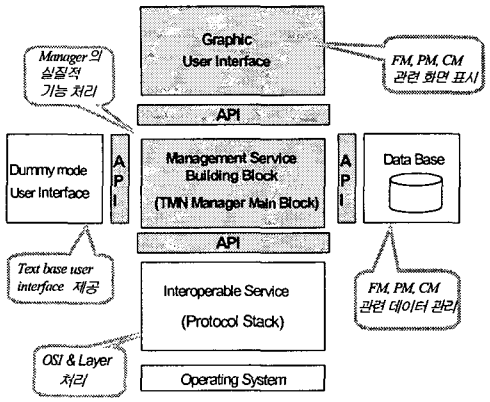

그림 2. 망 관리 매니저 소프트웨어의 구성도

CMIP 기반의 망 관리 매니저의 기능을 망 관리 관점의 계층적 구조로 나타내면 그림 3의 관리정보트리(MIT : Management Information Tree)와 같고, 이 논문에서는 EFD(EventForwarding Discriminator)와 equipment 관련 관리객체(MO : Managed Object)의 재사용성에 대하여 기술한다.

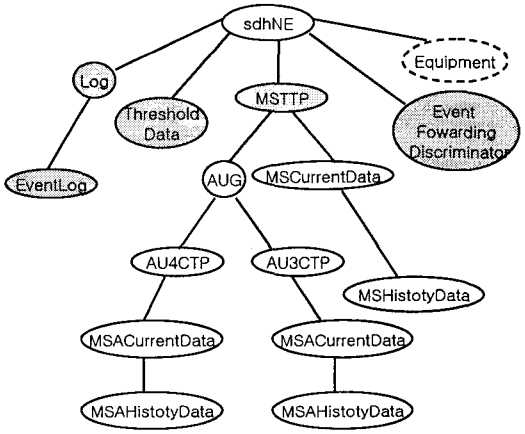

그림 3. 10Gb/s MIT 구조

EFD는 사건보고를 위한 관리객체로 사건보고에 대한 조건 및 사건보고 허용 여부 등을 나타내므로 시스템과 무관하게 구현된 내용을 재사용할 수 있다.

equipment는 시스템의 형상 관리를 위한 관리 객체로 euqipmentHolder는 랙, 셀프, 슬랏을 표현하고 있고 이 템플릿에 포함되어 있는 equipmentId는 이들 각각 인스턴스를 나타내어 주고 있다. 따라서 인스턴스 실체를 인식할 수 있는 RDN체계 분석 지식을 갖고 있어도 시스템의 형상이 다르므로 이를 그대로 이용하는 것은 매우 어렵다. 이러한 관점에서 CMIP 기반의 망 관리 매니저 소프트웨어를 DSC에 적용하면 그림 4와 같다.

```
Template SDH TMN MAnager software
begin
    parameters
        begin
            sorts FM, PM, CM
        end
                                //FM : Fault Management,
                            //PM : Performance Management,
                            //CM : Configuration Management
    exports
        begin
            sorts EFD
                    //EFD : Event Forwarding Discriminator
            functions event report
        end

    functions
        system configuration
    equations
        동기식 전송장치 기반의 Network
          Management
        Manager function
end
```

그림 4. 망 관리 매니저 소프트웨어에 대한 DSC

소프트웨어의 재사용 관점은 망 관리 매니저와 그래픽 사용자 인터페이스 사이의 접근을 위한 참조점(API : Access Point Interface)에서 CMIP 메시지를 분석하는 부분과 디코딩/인코딩하는 부분에 대하여 공통적으로 사용 가능한 기능과 시스템에 국한되어 새로 변경 또는 추가되는 부분에 대한 정의이다. EFD관련 소프트웨어는 현재 적용할 시스템에서 사용하는 RDN 체계에 따라 인스턴스에 대한 이름 값을 주고 그 외의 소프트웨어 전반적인 기능 그룹들은 그대로 사용한다. 즉, DSC에서 정의하는 exports, imports 부분은 재사용 되고, functions 부분은 관리대상이 되는 시스템에 맞도록 대부분의 스킴을 재구성하여 운용하도록 한다.

4.2 망 관리 매니저에 대한 평가

이 절에서는 DSC를 이용하여 명세화된 소프트웨어의 재사용성을 평가한다. 그림 3의 MIT에서 재사용 객체는 Log, EventLog, ThresholdData, Event Forwarding Discriminator이고, 수정객체는 MSTTP, AUG, MSCurrentData, AU4CTP, AU3CTP, MSHistoryData, MSACurrentData, MSAHistoryData이다. 또한 Equipment는 완전히 새로 개발하여야 하는 객체이다. 수정객체는 기본구조는 재사용되고 RR(Real Resource)와의 인터페이스 부분이나 또는 RR 인터페이

스는 없지만 시스템 자체의 고유한 특성을 나타내는 부분을 새로 구현하여야 한다.

소프트웨어 개발 관점에서 소요되는 노력의 구성 비율은 통계적으로 요구 사항 도출과 기본 구조의 구현에 50% 정도의 노력이 소요되고, 기본 구조에 RR 인터페이스 구현 부분이 나머지 50% 정도의 노력이 소요된다. 또한, 통신 소프트웨어는 자체 기능과 인터페이스가 중요하므로 실험에서는 이 요소만 고려하고 통신 소프트웨어 코드에 대한 테스트는 고려하지 않았다.

표 2. 10Gb/s MIT 노드

MIT 노드	재사용 수준		
	재사용 객체	수정 객체	구현 객체
Log	√		
ThresholdData	√		
MSTTP		√	
Equipment			√
EventLog	√		
AUG		√	
MSCurrentData		√	
Event Forwarding Discriminator	√		
AU4CTP		√	
AU3CTP		√	
MSHistoryData		√	
MSACurrentData		√	
MSAHistoryData		√	
TOTAL	0	400	100

이런 관점에서 완전 재사용 가능한 객체는 수정되거나 새로 개발할 필요가 없으므로 객체 당 개발비용의 가중치를 0으로 가정한다. 수정객체는 기본구조와 구현은 재사용이 가능하고 RR 부분은 새로 개발하므로 객체 당 개발비용의 가중치를 50으로 한다. 또한, 재사용이 불가능하여 새로 개발하여야 하는 객체는 객체 당 개발비용의 가중치를 100으로 가정한다. 시스템 S의 재사용성은 식 1과 같은 비율로 추정할 수 있다.

$$Rb(S) = [Cnoreuse - Creuse] / Cnoreuse, \quad 0 \le Rb(S) \le 1 \quad \cdots (식\ 1)$$

이 때, $Cnoreuse$는 재사용 없이 시스템 S를 개발하는 비용이고 $Creuse$는 재사용을 하여 S를 개발하는 비용이다. 표 2는 그림 3의 MIT 노드들의 재사용 정도를 측정한 것이다. 노드들의 총 개발비용 $Creuse$ = 500이다. 재사용 방법을 적용하지 않는 경우, 모든 노드들에 가중치를 100으로 하여 측정된 개발비용 $Cnoreuse$ = 1300에 비해 $Rb(S)$ =0.61의 재사용성을 보인다.

V. 결론

현재 컴포넌트의 모델링에 대한 관심이 높아지고 적용범위가 증가하고 있지만, 구체적인 지침이 마련되지 않는 상황에서 도메인내의 어플리케이션 개발에 기존에 구축된 모델을 재사용할 수 있다면 생산성과 품질 향상에 많은 도움을 줄 수 있을 것이다. 이 논문에서는 요구사항 정보나 설계 정보를 재사용하기 위한 도메인 명세화 컴포넌트(DSC)를 제안하였다. DSC는 컴포넌트를 단위로 하고, 비정형적인 명세화 기법이 갖는 애매 모호함과 정형적인 명세화 기법이 갖는 명세의 기술과 이해의 어려움에 대한 문제들을 보완하면서, 문제 도메인의 관점을 지원한다. 또한 객체 지향 소프트웨어의 특성인 추상화, 상속화, 정보 은닉, 일반화를 제공한다. 제안된 DSC 구조를 망 관리 매니저 소프트웨어와 적용한 결과 시스템의 특성을 정확히 명세화함을 확인하였고, 재사용성의 측정을 통해 시스템의 총 개발비용이 절감될 수 있음을 확인하였다.

향후 연구 과제로는 DSC를 이용하여 작성된 명세서가 요구 사항을 충족하는지를 검증할 수 있는 자동화된 검증 도구의 개발에 대한 연구가 요구된다.

참고문헌

[1] I. Somerville, *Software Engineering*, Addison-Wesley, 1995.

[2] Roger Duke, "Integrating Formal Methods with Object-oriented Software Engineering," *Proceeding Joint Conference on Software Engineering'93*, pp.3-10, 1993.

[3] Biggerstaf, T. and C. Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, 4(2), pp.41-49, Mar. 1987.

[4] Don Batory, David McAllester, Will Tracz, "Domain Modeling in Engineering Computer-Based Systems," *Workshop on Systems Engineering of Computer Based Systems*, 1995.

[5] Ibrahim R. and Szyperski C., "Formalization of Component Object Model (COM) - The COMEL Language," phD Workshop, *ECOOP'98*, 1998.