컴포넌트의 다면성과 서비스를 기반으로 하는 재사용 모델

(A Reuse Model Utilizing Diverse Aspects of Components and Services)

박 수 진 † 박 수 용 ††
(Soojin Park) (Sooyong Park)

요 약 소프트웨어 재사용을 위해 여러 가지의 접근법이 제시되어 왔으나, 소프트웨어 개발자들은 이미 개발된 소프트웨어 컴포넌트를 재사용하는데 있어서 여전히 회의적이다. 여러 가지 원인이 있겠으나, 기존의 재사용 접근법들이 개발자들이 재사용에 기울이는 노력을 감소시키는데 실질적인 도움을 주지는 못한 것이 가장 큰 원인 중 하나라고 볼 수 있다. 이러한 문제점 해결을 위해 본 논문에서는 RAS기반의 명세를 중심으로 컴포넌트의 다양한 측면을 활용하는 재사용 모델을 제시한다. 제시된 재사용 모델의 실용성 검증을 위해 실제적인 사례연구를 진행하였으며, 기존의 소스코드 기반의 소프트웨어 재사용 프로세스와 비교한 실험을 통해 재사용의 효율성 증가를 검증하였다. 이러한 검증을 통해, 제안된 재사용 모델이 이미 개발된 기존 컴포넌트를 개발자들이 검색하고 이해하는데 소요되었던 시간을 감소시킴으로써 전반적인 재사용 소요비용 감소시키고 있음을 확인할 수 있었으며, 제품라인공학과 같은 접근방법과 비교하여 일상적인 개발 공정의 지연을 초래할 수 있는 초기 투자비용을 최소화함에 따라, 자발적인 개발자들의 참여를 끌어낼 수 있다는 점에서 차별성을 찾을 수 있다. 뿐만 아니라, 제안된 재사용 모델은 컴포넌트 기반 개발 방법론이나 제품라인공학과 같은 기존의 재사용 방법들과 배타적인 개념이 아니므로, 필요에 따라 함께 적용하여 재사용 효과를 배가시킬 수 있을 것으로 기대한다.

키워드 : 소프트웨어 재사용, 소프트웨어 컴포넌트, 서비스 기반의 아키텍처, RAS

Abstract Even though many approaches for reuse have been introduced, software engineers are still hesitating to reuse existing software components. Among various reasons for the phenomena, the most significant one is that existing approaches failed to give substantial benefit for the reduction of developers' effort in reusing software assets. To solve this problem, we introduce a custom reuse model utilizing diverse aspects of components specified by RAS and services oriented architecture. We also carried out a case study to demonstrate its feasibility and evaluated it by comparing it to an existing code-based software reuse process. The proposed reuse model helps in the reduction of effort in reusing existing components by decreasing the time for searching and understanding them. Compared to other approaches such as product line engineering, our approach for software reuse using MPC does not require much initial work for implementing the reuse model in different projects. It is of interest to software engineers who are worried about heavy investment, which can cause the delay in their usual development work. Furthermore, the proposed reuse model is not mutually exclusive with other approaches for software reuse such as CBSD or product line engineering. It can accelerate the benefits gained from them.

Key words : Software Reuse, Component Based Software Development, Service Oriented Architecture, RAS

· 본 연구는 정보통신부 지원 ITRC 프로그램의 지원을 받아 수행되었음
† 학생회원 : 서강대학교 컴퓨터공학과
psjdream@gmail.com
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
sypark@sogang.ac.kr
논문접수 : 2006년 9월 1일
심사완료 : 2007년 1월 27일

1. 서 론

소프트웨어 규모의 대형화에 따라, 소프트웨어 개발환경은 더 복잡해지고, 개발주기는 더 길어지는 양상을 보이고 있다. 반면, 소프트웨어 자체를 사업기반으로 하는 비즈니스들의 증가로 인해, 시장적시성(Time-to-market)

은 성공적인 소프트웨어 개발이 반드시 만족시켜야 할 덕목이 되고 있다. 즉, 지금의 소프트웨어 개발 조직은 과거와 비교하여 더 복잡한 소프트웨어를 더 짧은 시간에 고품질을 유지하면서 개발해 내야 하는 요구사항을 만족시켜야 한다. 더 복잡한 소프트웨어를 더 짧은 시간에 개발해 내기 위해서는 이미 검증된 소프트웨어 자산의 재사용을 통한 생산성 증대가 하나의 대안이 될 수 있다[1].

비용 절감 이외에 소프트웨어 재사용이 표방하고 있는 궁극적인 목적은 소프트웨어 어플리케이션을 기존 소프트웨어 컴포넌트들의 조립을 통해 개발하는 데 있다. 이러한 목적 실현을 위해 현재까지 여러 가지 접근법들이 소개되어 적용되어 왔음에도 불구하고, 소프트웨어 개발자들은 여전히 이러한 접근법들이 자신의 노고를 대단히 감소시켜 줄 수 있다거나, 컴포넌트의 조합만으로 소프트웨어를 개발할 수 있다는 주장에 대해서는 매우 회의적이다.

현재까지 소프트웨어 재사용을 위해 적용된 접근방법 중 가장 보편적인 것으로는 컴포넌트 기반의 소프트웨어 개발 방법(Component Based Software Development, 이하 CBSD)을 들 수 있다. CBSD는 소프트웨어 시스템의 많은 부분들이 충분한 공통성(Commonality)을 가지고 있다는 가정에서부터 비롯되었다. CBSD의 적용으로 인해, 기존의 프로그래밍 위주의 소프트웨어 구축이 아닌 컴포넌트의 구성 혹은 조립 위주의 소프트웨어 구축으로 소프트웨어 개발의 초점은 이동되었다[2]. 그러나, 현재까지 시도되어온 CBSD 계열의 접근방법들은 다음과 같은 두 가지 측면에서 재사용 가능하고 교체 가능한 소프트웨어 아키텍처를 구성하는데 있어서는 충분한 해결방안을 제시하지 못하였다.

- 첫째, 컴포넌트 구축시 설계 단계에서 설정한 가정들을 자료화하여 컴포넌트에 탑재시키는 메커니즘을 제공하지 못하였다. 이로 인해 여러 가지 아키텍처 측면의 불일치가 설계단계나 구현단계에서 발생해 왔다[3].
- 둘째, 컴포넌트 소비자 측면에서 필요로 하는 컴포넌트를 획득한 경우라 하더라도, 해당 컴포넌트를 활용하기 위해 필요한 정보들이 누락된 채, 단지 실행 가능한 컴포넌트 혹은 소스코드 덩어리들만이 제공되는 경우가 많았다. 이는 컴포넌트 활용에 필요한 정보와 컴포넌트 자체가 분리되어 문서화되거나 관리되는 데서 기인한 문제로, 결과적으로 이미 요구사항을 만족시킬 만한 기존 컴포넌트가 확보되었음에도 불구하고 개발자들이 그것의 재사용을 꺼려하는 이유가 되기도 한다.

이러한 현상은 기존의 CBSD 관련 연구가 이미 제작된 컴포넌트들을 효과적으로 재사용하도록 하는 방안보다는, 어떻게 하면 컴포넌트를 잘 만들어낼 것인가의 문제에 집중되어 왔기 때문이다. 이에 반해, 본 논문은 컴포넌트의 효과적인 재사용 방안을 제시하는 데 그 목적을 두고 있다. 현재까지의 CBSD 기반의 기법들의 미비점을 보완하기 위해, 본 논문에서는 컴포넌트의 다양한 측면을 활용하는 재사용 모델을 제시하고자 한다. 본 재사용 모델에서는 RAS(Reusable Asset Specification, 이하 RAS)[4]와 SOA(Service Oriented Architecture, 이하 SOA)[5]를 적용하고 있다. RAS는 대상 소프트웨어 컴포넌트와 관련된 산출물로는 어떠한 것이 있으며, 실제 소프트웨어 어플리케이션 구축에 컴포넌트를 활용하기 위해서는 어떤 작업이 필요한 지 등에 대한 정보를 명시적으로 명세화함으로써, 소프트웨어 개발자간의 정형적인 의사소통을 도와주는 역할을 담당한다. 이렇게 RAS를 이용한 소프트웨어 컴포넌트 명세가 포함하고 있는 다양한 정보는 다시 온톨로지 모델로 전환, 관리됨으로써 효과적인 컴포넌트 검색 메커니즘을 제공하게 된다.

RAS를 이용한 컴포넌트 명세시 제기될 수 있는 이슈사항 중 하나는 하나의 RAS로 묶여지는 컴포넌트의 크기를 어떻게 정의하는가의 문제이다. 본 재사용 모델은 SOA의 개념을 기반으로 하여 컴포넌트의 크기와 관련된 이슈를 해결하고 있다. SOA에 대한 정의는 여러 가지가 있겠으나, "SOA는 자동화 가능한 로직이 하나의 좀 더 작고, 의미적인 측면에서 뚜렷한 특징을 가지는 로직의 단위로서의 모델로 표현하고 있다[5]."와 같이 다분히 추상적이어서, 개인별 해석의 상이함으로 인해 잠재적인 문제점을 야기할 수 있다. 따라서, 본 논문에서는 서비스(Service)를 "사용자의 관심 사항을 반영하는 하나의 워크플로우"로 정의하고, 유스케이스 명세서상에 나타나는 각각의 이벤트 흐름을 하나의 단일 서비스로 식별한 후, 실제 시스템이 고객에게 특정 서비스를 제공하기까지 필요한 모든 소프트웨어 자산들에 대한 정보를 하나의 RAS로 묶어 저장, 관리, 검색, 활용하도록 하였다.

기존의 컴포넌트를 기반으로 하는 재사용 관련 연구와 비교하여 본 논문의 차별성은 신규 생성 코드 대비 재사용된 코드의 비율만을 증가시킴으로써 재사용을 증대시키는 것이 아니라, 비록 동일한 정도의 코드를 재사용 한다 하더라도 효과적인 재사용 매커니즘을 통해 재사용에 소요되는 비용을 감소시킴으로써 전반적인 생산성을 향상시키고자 하는 데 있다. 이러한 실질적인 생산성 향상이 소프트웨어 개발자들이 기존에 가져왔던 소프트웨어 재사용에 대한 회의적인 태도를 완화시키는 데 도움이 될 수 있을 것이다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다.

2장에서는 현재까지 소개된 소프트웨어 재사용 모델과 비교하여 본 논문에서 제시하는 재사용 모델의 장점을 설명하고, 3장에서는 제안된 재사용 모델의 각 요소들을 설명한 후, 4장에서 실 프로젝트의 적용 사례를 소개한다. 5장에서는 제안된 재사용 모델의 실효성 검증을 위해 실행한 실험결과들을 소개하고 있다. 끝으로 6장에서 연구 성과를 종합한 후, 향후 연구에 대해 소개하고 있다.

2. 소프트웨어 재사용 모델

소프트웨어 재사용과 소프트웨어 개발 프로세스간의 연관관계는 두 가지의 관점에서 분석 가능하다. 첫 번째는 '재사용을 이용한 개발' 관점이고 두 번째는 ' 재사용을 위한 개발'이다[6]. 두 가지 관점 중 전자가 소프트웨어 개발자의 관점이라면, 후자는 소프트웨어 아키텍트의 관점으로 볼 수 있다. 재사용 개념을 실제 소프트웨어 개발 프로젝트에 적용하는 데 있어서 이 두 가지 관점 사이에는 좁혀지지 않는 견해 차이가 존재한다. 즉, 개발자의 관점에서 재사용 가능한 소프트웨어 자산은 이해하기 쉽게 작성되어 바로 재사용될 수 있는 형태의 비교적 적은 숫자의 컴포넌트를 의미하는데, 이것은 재사용 자산의 검색과 커스터마이징에 소요되는 비용을 최소화하는 것이 개발자들이 원하는 바이기 때문이다. 반면, 소프트웨어 아키텍트의 관점에서 바라보는 재사용 자산은 광범위하게 파라미터를 조정할 수 있는 방대한 양의 컴포넌트 집합이다. 왜냐하면, 아키텍트들의 최대 관심사는 시스템의 유연성이기 때문이다.

이러한 두 가지 상반된 재사용에 대한 관점간의 차이를 줄이기 위해서 다양한 형태의 재사용 모델이 제시되어 왔는데, [7]에서는 효율적인 소프트웨어 재사용을 달성하기 위해서는 프로젝트에 적용되는 여러 가지 측면에서의 재사용 모델들의 조화가 중요함을 설명하고 있다. 부합되는 모델을 찾기에 앞서, 소프트웨어 재사용 모델을 조직측면의 모델(Organizational model), 생산 측면의 모델(Production model), 투자 및 비용 관리 방법(Funding and Cost Management)과 같은 세 가지 범주로 나누어 각각의 범주에 속하는 모델들을 조사하고 이들간의 조합을 비교 분석하고 있는데, 표 1은 그 분석의 결과를 보여준다. 표 1에 따르면, 가장 바람직한 모델간의 조합으로 조직측면에서는 Expert Service Model 또는 Team Producer Model과, 생산 측면에서의 Post-project Generation Model 또는 Next Project Generalization Model, 그리고 투자 및 비용관리 방법으로 ABC 또는 Reuse Tax 방식간의 조합을 꼽고 있다.

본 연구에서 제안하는 MPC 기반의 재사용 모델은 투자 및 비용관리 방법은 그 범위에 해당되지 않으나, 나머지 두 가지 범주에 있어서 Expert Service 모델과 Post-project Generalization 기법을 사용하고 있다. 즉, 재사용 소프트웨어 컴포넌트의 검색 및 저장 관리를 전담하는 역할을 프로세스 상에서 식별하여 재사용 작업에 있어서의 전문가 역할을 담당하도록 하였으며, 미리 재사용 가능한 소프트웨어 자산의 범위와 존재를 가늠함에 따라, 프로젝트 착수 이전에 대규모의 노력을 요구하는 Pre-project Domain Analysis 기법이 아닌, 소프트웨어 컴포넌트 작성 및 테스트를 거친 이후 재사용 가능성이 있다고 판단되는 자산이 있을 때마다 별도의 메커니즘을 적용하여 재사용 컴포넌트 자산으로 등록 관리해 나가는 Post-project Generalization 기법을 채택하고 있다. 특히, 재사용 전문가가 활용 가능한 각 컴포넌트의 의미론적(Semantic) 측면의 정보들을 휘쳐그래프나 온톨로지 모델로 변환 관리함으로써, 재사용 전문가 역할로 식별된 MPC 등록자의 역할을 시스템화할 수 있는 가능성을 열어두고 있다는 점에서 [7]에서 소개하는 각 범주별로 규정지어질 수 있는 단편적인 재사용 모델과 비교했을 때, 소프트웨어 컴포넌트의 다각적인 측면을 활용한다는 장점이 있다.

3. MPC 기반의 재사용 모델

3.1 MPC(Multi-Phased Component)

MPC는 멀티페이스 컴포넌트(Multi-Phased Component)의 줄임말이다. 명칭에서 나타내고 있는 바와 같이,

표 1 체계적인 재사용을 위한 모델조합 평가

구분	가장 바람직한 조합	연간관계가 없는 조합	바람직하지 못한 조합
조직 측면의 모델	Expert Service Team Producer	Curator Model	Library, Product Center, Reuse Factory
생산측면의 모델	Post-project Generalization, Next-project Generalization	Pre-project Domain Analysis, In Project Domain Analysis/Generalization	Pre-project Domain Analysis, In Project Domain Analysis/Generalization
투자 및 비용관리 방법	ABC, Reuse Tax	Overhead, Pay-for-components	Pay-for-components

MPC는 컴포넌트를 구현하고 있는 소스코드뿐만이 아니라 해당 컴포넌트와 관련된 모든 개발 작업의 결과물을 함께 내장하고 있는 컴포넌트에 대한 명세서로 본 논문에서 제안하는 재사용 모델에서의 핵심적인 부분을 차지하고 있다. MPC는 OMG(Object Management Group)가 제정한 RAS(Reusable Asset Specification) [4]를 확장한 형태로, MPC에 대해 설명하기에 앞서, MPC의 모체가 된RAS에 대해 잠시 살펴보면 다음과 같다.

RAS는 OMG에 의해 채택된 컴포넌트 명세 표준으로, 동일 문맥상에서의 문제에 대한 해결책을 제시할 수 있는 재사용 가능한 소프트웨어 자산 집합체에 대한 명세이다. RAS는 소프트웨어 자산의 정확한 명세를 위해 자산 기반의 소프트웨어 개발과 관련 있는 표준 용어들과 소프트웨어 자산을 정의하기 위한 메타 모델 집합을 제공하고 있다. RAS는, 코어 RAS(Core RAS)와 프로파일(Profile), 두 가지 큰 범주로 나뉘어질 수 있다. 코어 RAS의 경우, 소프트웨어 자산 명세를 위한 기본적인 요소들을 표현하고 있다. 다음은 코어 RAS를 구성하고 있는 각 주요 섹션들에 대한 간략한 설명이다.

- 분류(Classification) 섹션: 소프트웨어 자산과 관련된 문맥(Context)에 대한 기술 및 자산을 분류하기 위한 기술자(Descriptor) 집합의 나열
- 솔루션(Solution) 섹션: 소프트웨어 자산에 포함된 산출물에 대한 기술
- 용법(Usage) 섹션: 소프트웨어 자산을 어떻게 설치하고, 자신의 어플리케이션에 맞게 변형시키고, 사용할 것인지를 기술
- 관련자산(Related assets) 섹션: 다른 소프트웨어 자산과의 연관성을 기술

위와 같이 하나의 소프트웨어 컴포넌트에 관련된 정보들이 여러 개의 섹션으로 나누어져서 재사용 가능한 소프트웨어 자산으로 명세화되는데 이때 사용되는 메타데이타들은 XML 스키마를 기반으로 하고 있으며, 각각의 자산 프로파일(Asset profile)들을 참조하게 된다. 현재, 여러 도구제공업체(Tool vendor)들이 RAS XML 스키마 파일을 사용하여, RAS의 실제 사용을 지원하고 있다. 실 사용시에는 코어 RAS 자체를 실체화(Instantiate)하여 사용하는 것이 아니라, 각자의 환경에 적합한 형태의 프로파일을 구성하여 적용하도록 하고 있는데, 본 논문에서는 효과적인 재사용을 위한 소프트웨어 컴포넌트 명세서로서, RAS의 기본 프로파일들 중에서 RAS 2.2. 버전의 디폴트 컴포넌트 프로파일을 확장한 하나의 프로파일을 정의하였고, 이를 멀티 페이스 컴포넌트(Multi-Phased Component)라 명명하였다.

기존의 RAS 2.2 버전의 코어 RAS에서 제공하는 클래스들은 상당히 높은 추상화 레벨에서 컴포넌트 관련 정보를 명세화하고 있다. 예를 들면, 다양한 소프트웨어 개발 단계로부터 작성된 산출물들을 명세하기 위한 클래스로 정의된 것은 'Artifact' 클래스 하나뿐이다. 여러 가지의 문맥 (Context)을 명시하는 방법으로 세분화 할 수는 있으나 직관적으로 각 산출물이 무엇인지를 파악하는 데는 다소 미비한 점이 있었다. 이러한 점을 보완하기 위해, OMG에서 제공하는 프로파일 들 중 하나인 디폴트 컴포넌트 프로파일에서는 기존의 'Artifact' 클래스를, 'Requirement.Artifact', 'Design.Artifact', 'Implementation.Artifact', 'Test.Artifact'와 같이 소프트웨어 개발 단계에 따라 분화시켰으나, 소프트웨어 개발의 다양한 측면을 표현하기엔 아직 충분하지 않은 점이 발견되었다. 예를 들면, 참조 아키텍처 모델(Reference architecture model)을 검색하고자 할 경우, 아키텍처와 관련된 자산이 별도로 식별될 수 있는 요소가 RAS 2.2의 디폴트 컴포넌트 프로파일엔 제공되지 않음에 따라, 모든 요구사항과 분석, 설계 자료들에 포함되어 있는 아키텍처 관련 자료들을 일일이 찾는 작업을 해야 한다.

이러한 점을 보완하여 MPC는 기존의 RAS와 비교하여 솔루션 섹션에 다양한 클래스들을 추가로 더 식별하거나, 기존 클래스들을 수정하였는데, 이는 그림 1에 짙게 표시된 클래스들과 동일하다.

기존의 RAS를 기반으로 MPC로 확장하는데 있어서 중점적으로 고려한 사항들은 아래와 같다.

- 아키텍처 중심의 소프트웨어 명세

아키텍처는 설계에 대한 근거가 될 수 있다는 점에서 소프트웨어 자산 중 중요한 부분을 차지한다. 아키텍처 관련 자산들의 효율적인 관리를 위해 MPC에서는 Architecture, Nonfunctional Requirement, Quality-Attribute와 같은 클래스들을 솔루션 섹션에 추가함에 따라 아키텍처와 관련된 부분들이 각 단계별 산출물에 내재되어 있는 것이 아니라 외부로 노출되어 직관적인 해석이 가능하게 되었다.

- 개념분리(Separation of concern)

RAS 2.2의 디폴트 컴포넌트 프로파일에서는 분석관련 산출물과 설계관련 산출물을 구분 없이 관리하도록 하고 있다. 설계 산출물로부터 분석 산출물을 분리할 경우, 분석모델 자체를 다른 구현 환경에 재사용할 수 있는 여지가 증가함에 따라 전반적인 재사용성의 증가를 도모할 수 있다는 점에서, MPC에서는 'Design' 클래스와는 별도로 'Analysis' 클래스를 식별하였다. 이러한 개념은 모델 주도적인 아키텍처(Model Driven Architecture)의 목적에도 부합된다.

- 가변점(Variation point)에 대한 명확한 명세

소프트웨어의 가변성을 문서화하는데 반드시 필요한

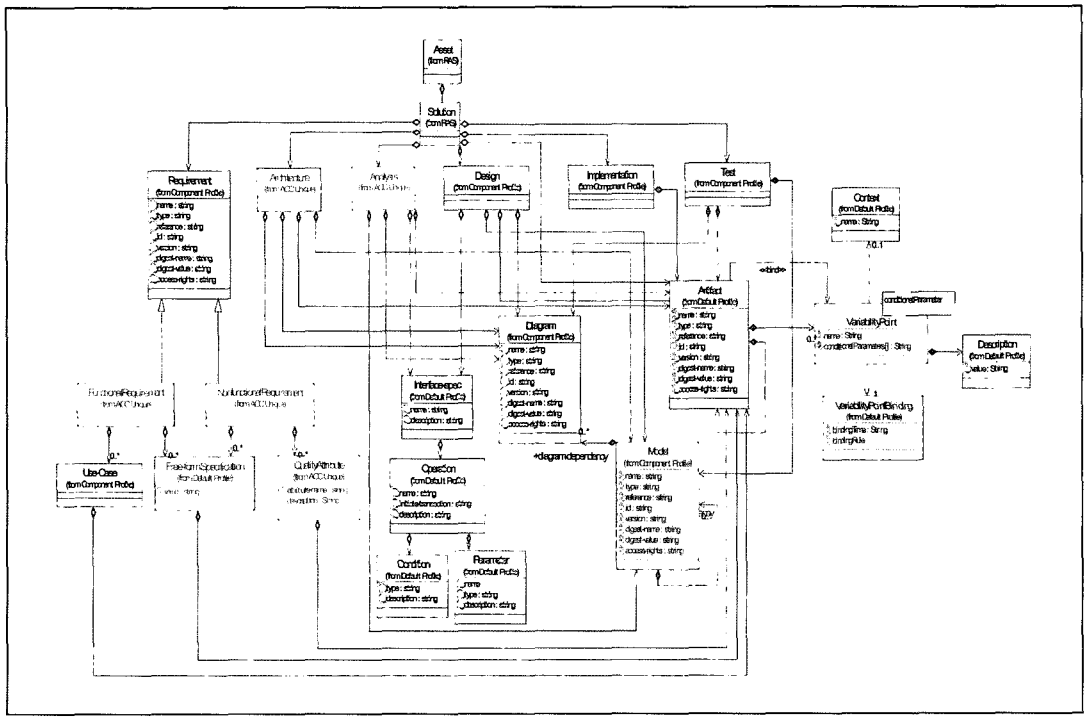


그림 1 MPC 솔루션 섹션의 주요 클래스

정보로는 가변점(Variation point), 옵션에 따라 영향을 받는 요소(Element)들, 그리고 각 옵션의 바인딩 타임(Binding time)이 있다[8]. 기존의 RAS에도 가변성을 명세하기 위한 클래스들이 제공되고 있으나, 변형점을 생성시키는 옵션을 명세할 수 있는 요소가 지원되지 않고 있었다. 이를 보완하기 위해, MPC에서는 'VariabilityPoint' 클래스를 파라미터화 클래스(Parameterized class)로 변경하여, '<<bind>>' 의존관계(Dependency)가 어떤 'Artifact' 클래스가 주어진 'VariabilityPoint'와 관련되어 있는 지를 명확히 나타낼 수 있도록 하였다.

솔루션 섹션 외에도 분류 섹션에서는 기존 RAS와 비교하여 효율적인 컴포넌트의 분류와 검색을 돕기 위해 여러 가지의 컨텍스트 카테고리를 추가하였는데, 아키텍처 스타일(Architecture style), 설계 패턴(Design pattern), 개발방법(Development method), 어플리케이션 영역(Application domain), 프로그래밍 언어(Programming language) 등이 그것이다. 이러한 컨텍스트 카테고리들 중 대부분은 MPC 검색시 검색어로 활용된다.

3.2 MPC를 활용한 재사용 프로세스 모델

본 논문에서 제안하는 재사용 프로세스 모델은 개개의 소프트웨어 컴포넌트 개발 방법을 다루고 있는 것이 아니라, MPC를 이용한 소프트웨어 재사용에 관한 프로세스에 대한 하나의 전형으로, SOA를 바탕으로 구성되었다. SOA는 소프트웨어를 서비스의 집합체로 바라보는 하나의 새로운 패러다임으로, 엔터프라이즈 어플리케이션 내부 혹은 어플리케이션들간의 분산 컴퓨팅 환경과 협업을 지원한다[9]. SOA는 서비스들에 관한 아키텍처일 뿐만 아니라 서비스와 관련된 세 가지 종류의 참여자들의 역할을 정의하고 있는데, 서비스 제공자(Service provider), 서비스 등록자(Service registry), 그리고 서비스 요청자(Service client)로 나뉠 수 있다. 이들간의 상호동작으로는 공포(Publish), 검색(Find), 결합(Bind) 등의 오퍼레이션이 있다[9].

MPC를 활용한 재사용 프로세스 모델은 재사용에 참여하는 역할자를 SOA에서 정의한 참여자의 역할(서비스 제공자/ 등록자/ 요청자)과 유사하게 MPC 제공자/ 등록자/ 요청자로 구분하고 있다. 그 외에도 작성된 소프트웨어 산출물들을 하나의 MPC로 패킹하는 시점에서 하나의 서비스를 패킹의 단위로 채택하고 있다는 것도 SOA의 특징적인 개념을 활용한 예이다.

그림 2에서 도식화 하고 있는 MPC를 활용한 재사용 프로세스 모델의 각 액티비티를 참여자별 역할에 따라 살펴보면 다음과 같다.

3.2.1 MPC 요청자(MPC Client)

MPC 요청자는 이미 개발된 소프트웨어 컴포넌트를 재사용함으로써 새로운 시스템의 일부분을 구축하는 역

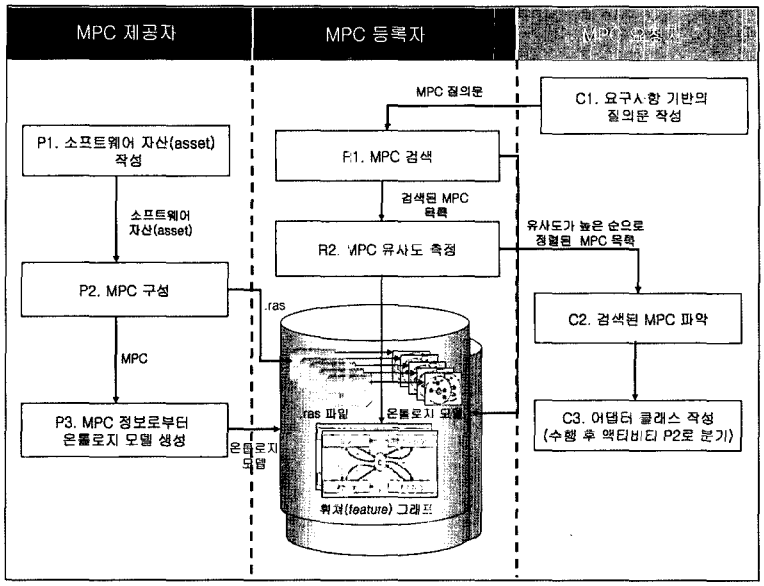


그림 2 재사용 모델의 액티비티 개관

할을 말한다.

C1. 요구사항 기반의 질의문 작성

MPC 요청자는 필요로 하는 컴포넌트에 대한 요구사항을 파악하여 주요 키워드(Keyword)를 추출한 후, MPC의 구조에 비추어 적절한 질의문을 작성한다. 각각의 단일 검색 질의문 사이에는 AND 관계가 성립하므로(그림 3(a) 참조), 입력된 모든 조건을 만족하는 MPC를 검색하게 된다. 요구사항으로부터 추출된 주요 키워드들은 MPC의 클래스들 중 주로 분류(Classification) 섹션에 속하는 클래스의 속성들과 비교함으로써 원하는 MPC를 검색하게 된다.

C2. 검색된 MPC 파악

MPC 요청자가 작성한 컴포넌트 검색을 위한 질의문은 검색을 위해 MPC 등록자에게 넘겨진다. MPC 등록자는 해당 질의문에 따라 MPC 저장소를 검색하여 유사도순으로 나열된 후보 컴포넌트 목록을 MPC요청자에게 다시 넘겨주게 된다. 본 모델에서 사용하고 있는 유사도 측정 기법에 따라, 검색된 후보 컴포넌트들이 유사도순으로 나열되어 주어지지만, 최종적으로 어플리케이션 개발에 사용될 컴포넌트를 선택하는 것은 MPC 요청자의 책임이다. 주어진 요구사항에 가장 근접한 MPC를 파악하는데 있어서 유용한 정보는 MPC를 이루는 요소들 중에서, 용법(Usage) 섹션, 해결책(Solution) 섹션 중 가변점(Variability point) 관련 부분과 인터페이스 사양(Interface-Spec)관련 부분, 그리고 분류(Classification) 섹션 등에 수록되어 있다.

C3. 어댑터(Adapter) 클래스 작성

C2 액티비티를 통해 선택된 재사용될 컴포넌트가 별도의 수정작업 없이도 요구사항을 만족시킬 수 있다면, C3 액티비티는 수행하지 않아도 된다. 그러나, 기개발된 컴포넌트를 재사용하지만, 새로운 시스템에서 요구하는 컴포넌트 요건을 충족시키기 위한 추가적인 구현이 필요할 경우, 하나 이상의 어댑터(Adapter) 클래스를 작성하여, 대상(Target) 컴포넌트를 구성한다. 어댑터 클래스 작성시의 설계전략은 [11]의 어댑터 패턴(Adapter pattern)을 바탕으로 한다. 즉, 위임(Delegation)과 특수화(Specialization)를 활용하여 기존 소프트웨어 컴포넌트의 자산을 재사용하게 된다.

이때 새롭게 개발된 어댑터 클래스가 향후 개발에서 재사용 가능성이 있다고 판단될 경우, MPC 제공자 역할에서의 P2액티비티로 프로세스 플로우가 분기되어 다시 새로운 MPC를 구성하게 된다.

3.2.2 MPC 제공자(MPC Provider)

MPC 제공자는 재사용 가능한 소프트웨어 자산을 공포(Publish)하는 역할이다.

P1. 소프트웨어 자산 작성

각각의 소프트웨어 자산들은 프로젝트마다 정의된 개발 프로세스에 의해 작성된다. 작성된 소프트웨어 자산들 중 향후 재사용의 가능성이 있다고 판단될 경우, 뒤따르는 P2 액티비티에서 각각의 서비스별로 묶여져서 하나의 MPC를 구성하게 된다.

P2. MPC 구성

MPC제공자는 새로운 소프트웨어 컴포넌트를 개발한 후 단위테스트를 거쳐 문제가 없을 경우, 해당 컴포넌트

와 관련된 모든 관련 산출물에 대한 정보를 하나의 MPC로 구성한다. MPC로 정보들을 패킹(Packing)할 때, 하나의 MPC에 수록되는 정보의 영역은 유스케이스 명세서(Use-Case specification)상에서 하나의 이벤트 흐름(Flow of event)으로 식별되는 단일 서비스를 기준으로 한다. 즉, 단일 서비스와 관련되어 있는 소프트웨어 아키텍처, 분석/설계 모델, 구현 클래스들과 테스트 케이스 등이 하나의 MPC에 수록된다. 이렇게 서비스 기준의 MPC구성이 적용 가능한 부분은 시스템의 어플리케이션(Application) 영역이다. 하나의 MPC에 수록된 정보들은 저장과 관리를 위해 하나의 저장파일(Archive file)형태로 압축되는데, 이때 저장파일의 형식은 '.ras'이다. '.ras'파일은 OMG가 정의한 압축파일 형태로 RAS를 지원하기 위한 것이다. MPC는 지원도구 사용에 있어서의 상호운영성(Interoperability)을 고려하여, OMG의 표준인 RAS의 저장 메커니즘을 그대로 따르고 있다.

P3. MPC모델로부터 온톨로지 모델 생성

MPC의 구조는 서비스 문맥을 기준으로 컴포넌트를 이해하기에 유용하도록 구성되어 있다. 그러나, 실제 컴포넌트를 시스템에 활용하기 위해 파악하는 단계에서는 하나의 MPC에 수록된 컴포넌트 내부의 개별적인 클래스들간의 관계까지도 이해하여야만 정확한 활용이 가능하다. 이러한 클래스 레벨의 정보들을 실제 설계모델을 열어보지 않고도 손쉽게 이해하도록 하기 위해, MPC에 수록된 UML모델의 내용은 1:1로 대응되는 온톨로지 모델(Ontology model)[12]로 전환된다. 온톨로지를 나타내는 여러 가지 지식표현법 중에서도 본 논문에서는 프레임 기반의 표현법(Frame Based Representation, 이하 FBR)[12]을 사용하고 있다. 이는 [13]에 의하면 FBR의 여러 요소들이 객체지향 개념과 매우 유사하며, 객체지향 프로그래밍의 요소들과 프레임 사이에는 1:1 대응관계가 성립되기 때문이다. 본 논문에서는 [13]에서 제시하는 매핑을 근간으로 클래스간의 관계와 프레임간의 관계의 매핑을 정리하여 다음과 같은 객체-프레임간의 전환 규칙을 사용하였다.

- Class name→Frame name
- Class type→Frame type
- <<generalize>>relationship→A-Kind-Of, Descendants
- <<aggregation>>relationship→Has-part
- <<association>>relationship→Semantic-link-from/ Semantic-link-to
- Attribute→Slot

하나의 MPC가 저장소에 저장될 때 새로이 작성된 어댑터 클래스의 생성으로 인해 파생된 새로운 클래스 혹은 컴포넌트들간의 관계는 휘처그래프(Feature graph)[14]에 의해 표현된다. 클래스 레벨의 휘처 그래프는 클래스간의 재사용 관계뿐만 아니라, 실제 개발과정에서 클래스간 재사용에 소요된 시간(Man-Hour)을 노드 사이의 연결선에 명시하고 있다(그림 3(c)참조). 클래스 레벨의 휘처 그래프는 MPC 저장 시에 함께 MPC저장소에 저장되어 후속 어플리케이션 개발 시 컴포넌트 재사용의 참고자료로 활용된다.

3.2.3 MPC 등록자(MPC Registry)

MPC 등록자는 MPC 요청자가 요청한 MPC들을 검색하고, MPC 저장소에 저장된 MPC 관련 정보들을 관리하는 역할이다.

R1. MPC 검색

컴포넌트가 포함하고 있는 의미들과 관련 소프트웨어 산출물들간의 연관관계 등은 MPC의 구조를 사용하여 의미론적인 검색이 가능하다. MPC의 구성요소에 대한 효과적인 검색을 위해 검색우선순위를 부여하였는데, 그 순서는 (1) 분류섹션(Classification section), (2) 'Solution.Design. Interface-Spec' 클래스, (3) 'Solution. Nonfunctional Requirement.QualityAttri bute' 클래스, 그리고 그 외 요소들 순이다. 이러한 검색 우선순위는 검색범위를 좁혀 나감으로써 검색의 총 횟수를 줄이기 위한 것이다.

R2. MPC 유사도 측정

검색을 통해 얻어진 MPC 목록에 속한 MPC들간의 유사도 측정은 근접관계(Closeness relation)[14]를 통해 계산 가능하다. 기존의 컴포넌트의 수정을 통해 새로운 컴포넌트를 구축한 경우, 둘 간의 근접관계(Closeness relation)는 휘처 그래프(Feature graph)를 사용하여 도식화한다. 휘처 그래프상의 노드는 하나의 개념(Concept)을 나타낸다. 하나의 컴포넌트라 하더라도, 여러 개의 개념이 적용될 수 있다. 예를 들어, 사례연구에서 구축대상이 되는 컴포넌트는 물종으로는 '의료물자(Medical Stuff)', 사용되는 개발 방법론은 'RUP', 소요산정 기준은 '장비(Equipment)', 그리고 구현언어로는 'C++'이 사용되는 개념(Concept)들로 규정지을 수 있다. 이러한 각각의 개념들은 각 휘처 그래프를 이루는 노드(Node) 형태로 표현된다. 근접관계는 이러한 휘처 그래프 상에서의 노드간의 연결선으로 표현되는데, 노드간의 상관관계 이외에 표현할 수 있는 정보로는 가중치(Weight)가 있다. 근접관계를 나타내는 연결선에 표시된 숫자는 화살표 시작방향에 연결된 개념에서 화살표 종료방향에 연결된 개념으로 이전하기까지 소요되는 노력정도(Effort)를 정량화한 것이다.

즉, 기존 컴포넌트를 수정하여 새로운 컴포넌트의 요구사항을 만족시키기까지 소요되는 노력 정도가 두 컴포넌트들간의 관계에 대한 가중치로 설정되어 표시된다.

이때 노력정도로 측정 가능한 값이 여러 가지가 있겠으나, 본 논문에서는 기존 컴포넌트 수정을 통해 새로운 컴포넌트를 구축하기까지 걸린 시간(Man-Hour)을 단위로 사용하였다.

그림 3(c)에서 'Food' 노드에서 'Commodity' 노드까지의 근접관계에 '31'이라고 표시된 것은 식료품목의 예산소요산정을 위한 컴포넌트를 활용하여 일반물자의 예산소요산정용 컴포넌트 구축시 31 Man-Hour가 소요되었다는 것을 의미하며, 이 수치는 선행 개발 프로젝트들로부터 축적된 경험치의 평균값이다.

휘쳐 그래프에서 중앙의 'Θ' 노드로부터 각각의 다른 노드로 나가는 근접관계는 기존 컴포넌트의 재사용없이 완전히 처음부터 개발했을 때에 소요되는 노력정도를 표시한다. 'Θ' 노드로 들어가는 근접관계가 가지는 가중치는 항상 '0'으로 고정되어 있다. 앞에서 설명한 휘쳐 그래프의 노드간 가중치를 기반으로 컴포넌트간 근접거리는 다음과 같은 식으로 계산된다. MPC요청자가 요구한 컴포넌트 'B'(개발대상인 컴포넌트)와 MPC저장소에서 검색된 이미 개발된 컴포넌트 'A'간의 근접거리 :

$$D_c(A,B) = \sum_{f \in \Psi} D_f(A.f, B.f)$$

- Ψ : 휘쳐공간(Feature space)
- $D_f(t_1,t_2)$: t_1에서 t_2까지의 최단경로값, 즉, 최단경로상의 가중치의 합
- D_c: 근접도 비교자(Closeness comparator)
- D_f: 휘쳐 비교자(Feature comparator)

4. 사례연구

3장에 기술한 재사용 모델이 실제 소프트웨어 개발에 어떻게 적용되는 지를 보이기 위해, 국방 물자관리 시스템을 대상으로 사례연구를 수행하였다. 사례연구의 대상인 군수 물자관리 시스템은 물종별로 물자의 소요 계산부터 유통, 사용, 소모처리에 이르기 까지를 지원하는 시스템으로, 4개의 회사에 속한 개발팀에서 개발이 진행되었다. 물종별로 나뉘어 개발되었으나 시스템이 지원하는 기능은 동일하다는 점에서, 상당히 많은 부분의 컴포넌트 재사용 가능성이 있는 소프트웨어 개발이었다. 그러나, 실제 개발 조직이 서로 다른 회사에 속한 개발팀으로 원활한 의사소통에 제약이 있음에 따라, 상당 부분의 중복개발이 이뤄지고 있었다.

국방 물자관리 시스템은 아키텍처 측면에서, 프리젠테이션 계층(Presentation layer), 비즈니스 프로세스 계층(Business process layer), 비즈니스 로직 계층(Business logic layer), 데이타 계층(Data layer)과 같은 4개의 계층(Layer)으로 설계되었으며, 전자의 두 계층은 어플리케이션 개발 영역에 속하며, 후자의 두 계층은 도메인(Domain) 엔지니어링 영역에 속한다. 컴포넌트 개발에 사용된 개발 방법론으로 RUP를 사용하고 있으며, 구현 언어로는 C++가 사용되었다.

현재 MPC기반의 재사용 모델을 지원하기 위한 자동화 환경은 구축 중에 있어서, 본 장에 소개되는 사례 연구의 결과물들은 모두 수작업에 의해 작성되었다. 실질적인 적용 과정과 결과를 설명하기에 앞서, 사례연구의 개발 시나리오를 설명하면 다음과 같다.

개발팀에게 '의료장비의 물자소요 예산산정' 이라는 새로운 서비스 개발 요청이 접수된다. 아키텍트는 다른 팀에서 이미 개발해 놓은 컴포넌트를 재사용하여 새로운 서비스를 시스템에 반영할 것을 개발자에게 지시한다. 현재, 의료품 외에 타물종의 물자보급 시스템을 개발하는 팀에서는 이미 '물자소요 산정'과 유사한 서비스를 구축해 놓은 상태이다.

위와 같은 개발시나리오를 가지고 3.2절에서 소개된 각 액티비티를 실제 국방 물자 관리 시스템 개발에 적용한 결과는 다음과 같다. 3.2절에서는 참여자 역할별로 액티비티를 기술하였으나, 사례연구에서는 프로세스의 흐름에 따라 각 액티비티들을 기술하도록 한다.

C1. 요구사항 기반의 질의문 작성

아키텍트가 개발자에게 할당한 컴포넌트 개발에 대한 주요 요구사항은 다음과 같다. "기능적인 측면에서, 익회계연도에 해당하는 의료물자의 소요를 산정하는 서비스를 구축하도록 한다. 비기능적인 측면에서 만족시켜야 할 품질속성은 유지보수성(Maintainability)과 보안성(Security)으로, 소요산정을 위한 규칙들이 변경되더라도 시스템이 이를 무리 없이 반영하도록 해야 하며, 오로지 인증된 사용자만이 예산소요 자료에 접근할 수 있도록 해야 한다. 예산 소요 자료의 수정은 일반적인 자료 접근 보다는 더 높은 정도의 보안 레벨을 취득한 사용자만이 수행할 수 있도록 해야 한다." 이러한 요구사항에 부합하는 컴포넌트를 검색하기 위해 작성된 질의문은 그림 3(a)와 같다.

R1. MPC 검색

그림 3(a)의 질의문을 가지고 MPC를 검색한 결과 검색된 후보 MPC들은 그림 3(b)의 표에서 보이는 바와 같이 BudgetMngnt, WartimeBudget, ComposeBudget, EstimateBudget 등이다. 그림 3(b)에서 가장 중앙열에 있는 '소요 산정기준'은 동일한 물종에 속한 품목들은 유사한 소요산정 기준을 바탕으로 예산소요를 산정한다는 점에서, 컴포넌트들간의 유사도 측정시에 가장 유용한 휘쳐이다. 그 외에도 물종, 사용된 개발방법론, 구현 언어 등의 휘쳐에 대한 각 컴포넌트의 속성값들을 그림 3(b)의 표에서 확인할 수 있다.

R2. MPC 유사도 측정

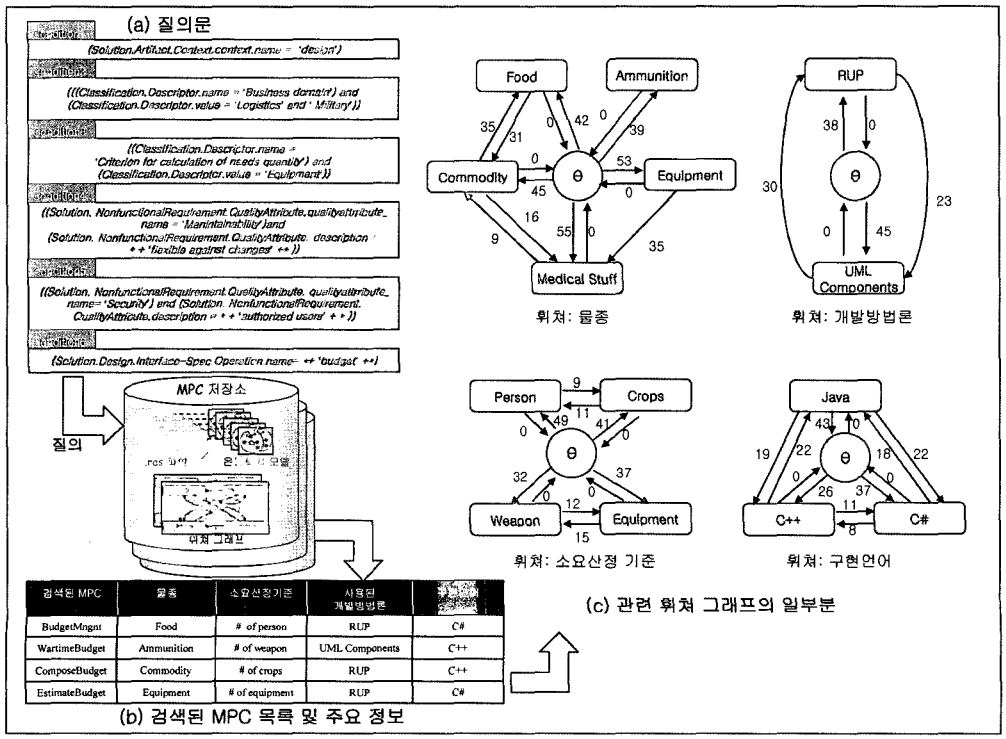
그림 3(c)의 휘쳐 그래프들로부터 우리가 개발하고자 하는 컴포넌트(Required component)와 3(b)에서 단어 매칭(Word matching)을 통해 검색된 각 컴포넌트들간의 근접도를 3.2절의 R2 액티비티에서 소개한 산정공식에 따라 산정한 결과는 아래와 같다. 그림 3(c)의 휘쳐 그래프가 나타내고 있듯이 유사도 계산에 사용된 휘쳐는 물종(Item category), 소요산정기준(Criterion for calculation of needs quantity), 개발방법론 (Development method), 구현 언어(Programming language), 네 가지이다.

- Ψ = {Item category, Criterion for calculation of needs quantity, Development method, Programming language}
- Dc(BudgetMngnt, required component)
  = Dcategory(Food, Medical Stuff)+Dcriterion(Crops, Equipment)+Dmethod(RUP, RUP)+Dpl(C#, C++)
  = 55 + 37 + 0 + 8 = 100
- Dc(WartimeBudget, required component)
  = Dcategory(Ammunition, Medical Stuff)+ Dcriterion(Weapon, Equipment) + Dmethod (UMLComponents, RUP)+ Dpl(C++, C++)
  = 55 + 12 + 30 + 0 = 97
- Dc(ComposeBudget, required component)
  = Dcategory(Commodity, Medical Stuff) + Dcriterion(Crops, Equipment) + Dmethod(RUP, RUP)+ Dpl(C++, C++)
  = 16 + 37+ 0 + 0 = 53
- Dc(EstimateBudget, required component)
  = Dcategory(Equipment, Medical Stuff)+ Dcriterion (Equipment, Equipment) + Dmethod(RUP, RUP) + Dpl(C#, C++)
  = 35 + 0+ 0 + 8 = 43

R2 액티비티의 최종 산출물 집합은 최근접 컴포넌트 외에도, 근접도에 따라 내림차순으로 정렬된 후보 컴포넌트들의 목록까지를 포함한다. 따라서, 본 사례연구에서는 최근접 컴포넌트인 'EstimateBudget'과 근접한 순으로 정렬된 후보 컴포넌트 목록(EstimateBudget→ComposeBudget→WartimeBudget→BudgetMngnt)이 최종 산출물이 된다.

C2. 검색된 MPC 파악

최근접 컴포넌트인 'EstimageBudget' MPC의 'Interface- Spec' 클래스에 수록된 정보를 검토한 결과, 'EstimageBudget' MPC는 우리가 원하는 컴포넌트의 모



검색된 MPC	물종	소요산정기준	사용된 개발방법론	구현 언어
BudgetMngnt	Food	# of person	RUP	C#
WartimeBudget	Ammunition	# of weapon	UML Components	C++
ComposeBudget	Commodity	# of crops	RUP	C++
EstimateBudget	Equipment	# of equipment	RUP	C#

그림 3 MPC의 정보를 이용한 컴포넌트 검색과 유사도 측정

든 기능들을 구현하고 있지는 않았다. 'EstimateBudget' MPC가 충족시키지 못하는 부분들을 다시 질의문으로 구성하여 적절한 MPC를 검색한 결과, 나머지 기능들을 구현할 수 있는 'ComposeBudget' MPC를 얻을 수 있었다. 다시 말해, 개발요청이 접수된 '의료 장비의 물자소요 예산산정' 서비스는 이미 개발된 두 개의 컴포넌트를 재사용함으로써 구축 가능하다는 사실을 알게 되었다.

C3. 어댑터(Adapter) 클래스 작성

앞선 스텝에서 획득한 두 개의 컴포넌트 중, 'Estimate-Budget' 컴포넌트는 특수화 관계 (<<specialize>> relationship) 설정을 통해, 'ComposeBudget' 컴포넌트와는 결합관계 (<<association>> relation-ship) 설정을 통해 필요한 오퍼레이션들을 재사용함으로써, 새로운 컴포넌트인 'DrawBudget' 을 구현할 수 있었다. 그림 4(b)는 이러한 'DrawBudget' 의 내부 클래스 다이어그램 중 일부를 보여주고 있는데, 'ComposeBudget'과 'Estimate-Budget'의 내부 클래스들을 활용하기 위한 어댑터 클래스인 'PlanBudget_8'이 새로이 정의되었음을 확인할 수 있다.

P2. MPC 구성

국방 물자관리 시스템의 아키텍처를 이루는 4개의 계층중 비즈니스 어플리케이션 계층과 비즈니스 프로세스 계층이 서비스 기준의 MPC구성이 가능한 어플리케이션 영역에 해당되며, 새로 작성된 'DrawBudget'은 이 영역에 속하는 컴포넌트이다. 따라서, 'DrawBudget' 컴포넌트와 관련된 모든 산출물들은 하나의 MPC로 압축되어 저장, 관리, 검색된다. 일반적으로 RAS가 수록한 정보를 실제 압축 파일인 '.ras'로 압축하는 데는 도구가 사용되는데, 본 사례연구에서는 IBM Rational Software Architect 6.0이 사용되었다. 그림 5는 'DrawBudget' MPC를 압축한 'DrawBudget.ras' 파일내에 수록된 모든 소프트웨어 산출물에 대한 목록인 'manifest.rmd' 의 내용을 보여주고 있다. 'manifest.rmd' 파일은 일종의 XML파일로 앞의 R1액티비티에서 MPC 검색 시 가장 먼저 검색하게 되는 파일이다.

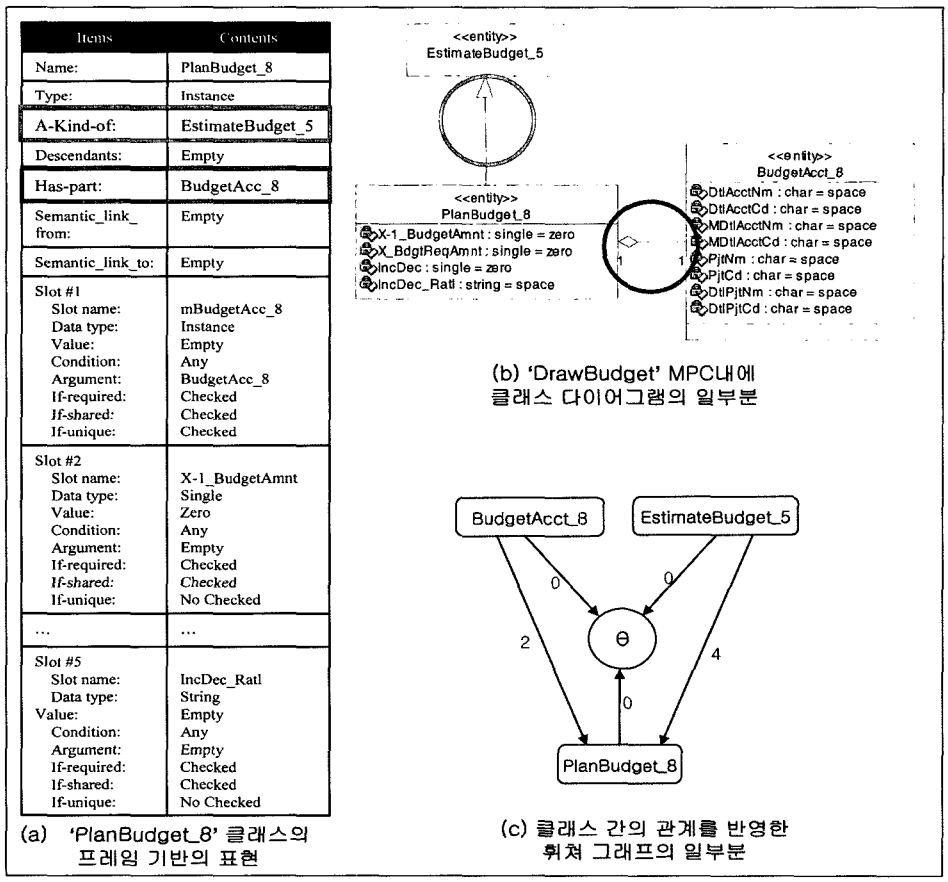


그림 4 UML 모델에 수록된 정보의 온톨로지로의 전환

```
<?xml version="1.0" encoding="UTF-8"?>
<defaultprofile:Asset xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:defaultprofile="http:///defaultprofile.ecore"
name="DrawBudget" id="{323DA7DD-6097-B672-B946-7C6D8FF65D61}" date="2006-07-04" version="1.0" shortDescription="DrawBugetMPC">
 <classification>
  <descriptorGroup name="Default">
   <descriptor name="Business Domain">Military Logistics</descriptor>
   <descriptor name="Architectural Style">Layer, Component&amp;amp;Connector</descriptor>
   <descriptor name="Design Pattern">Adapter, Proxy</descriptor>
   <descriptor name="Development Method">RUP</descriptor>
   <descriptor name="Application Domain">Enterprise Application</descriptor>
   <descriptor name="Programming Language">C++</descriptor>
   <descriptor name="MPC Provider">Soojin Park</descriptor>
  </descriptorGroup>
 </classification>
 <solution>
  <artifact name="InterfaceRequirementSpecification.doc" type="Microsoft Word" reference="./Analysis/InterfaceRequirementSpecification.doc">
   <artifact name="CalcBudgetNeed" type="Operation">
    <artifact name="Target Year" type="Parameter"/>
    <artifact name="Item Code" type="Parameter"/>
    <artifact name="Budget account code" type="Parameter"/>
   </artifact>
   <artifactType type="Analysis.interface-spec"/>
  </artifact>
  <artifact name="SoftwareArchitectureDocument.doc" type="Microsoft Word" reference="./Architecture/SoftwareArchitectureDocument.doc">
   <artifactType type="Architecture.Artifact"/>
  </artifact>
  <artifact name="Guideline for Deployment.doc" type="Microsoft Word" reference="./Guideline for Deployment.doc"/>
  <artifact name="Guideline for Oracle Environment.doc" type="Microsoft Word" reference="./Guideline for Oracle Environment.doc"/>
  <artifact name="BudgetAcct_8.cpp" type="CPP" reference="./Implementation/BudgetAcct_8.cpp">
   <variabilityPoint name="ItemCategory"/>
   <artifactType type="Implementation.Artifact"/>
  </artifact>
  <artifact name="BudgetAcct_8.h" type="C Header" reference="./Implementation/BudgetAcct_8.h">
   <artifactType type="Implementation.Artifact"/>
  </artifact>
   ----------omitted----------
 </solution>
 <usage>
  <artifactActivity>
   <activity task="Draw Budget according to the item" role="Calculation" taskType="CPP" reference="//@solution/@artifact.4">
    <variabilityPointBinding bindingRule="If itemcode is started with '8e', predefined classe(s) are set as active in the system."/>
   </activity>
  </artifactActivity>
  <assetActivity>
   <activity task="Deploy" role="Deployment manager" taskType="Deploy developed components" reference="//@solution/@artifact.2"/>
  </assetActivity>
  <assetActivity>
   <activity task="Environment" role="Tool specialist" taskType="Set up Oracle DBMS" reference="//@solution/@artifact.3"/>
  </assetActivity>
 </usage>
 <relatedAsset name="ComposeBudget" relationshipType="dependency" assetId="Rel_01" reference="//@solution/@artifact.16">
  <description>This relationship comes from the aggregation between PlanBudget_8 in this MPC and BudgetAcct_8.</description>
 </relatedAsset>
 <relatedAsset name="EstimateBudget" relationshipType="parent" assetId="Rel_02" reference="//@solution/@artifact.15">
  <description>This relationship comes from the inheritance between PlanBudget_8 in this MPC and EstimateBudget_5.</description>
 </relatedAsset>
 <profile name="Default Profile" idHistory="F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F"
  versionMajor="2" versionMinor="11">
  <description>The Default Profile is a realization of the Core RAS metamodel as defined in the OMG Reusable Asset Specification.</description>
 </profile>
 <description>MPC to draw up an estimate for next years with various needs quantity on medical stuff. The calculation metric of estimated budget for
  medical stuff is 'Equipment'. From the perspective of non-functional aspect, the component is related to 'maintainability' and 'security'.</description>
</defaultprofile:Asset>
```

그림 5 'DrawBudget.ras'의 'manifest.rmd'

P3. MPC모델로부터 온톨로지 모델 생성

그림 4(a)는 'DrawBudget' 컴포넌트 개발시 신규 작성된 어댑터 클래스인 'PlanBudget_8'(그림 4(b))로부터 온톨로지 모델을 구성하는 프레임으로 전환된 모습을 보여주고 있으며, 그림 4(c)는 새로 개발된 'DrawBudget' MPC 내부에 새로운 어댑터 클래스인 'PlanBudget_8'이 생성되었고, 'EstimateBudget' 컴포넌트와 'Compose-Budget' 컴포넌트에 속한 'EstimateBudget_5' 클래스와 'BudgetAcct_8' 클래스를 재사용하여 요구되는 서비스를 구현하고 있음을 보여주는 클래스 레벨의 휘쳐그래

프를 보여준다. 앞서 3.2절에서 설명한 바와 같이, 클래스 레벨의 재사용과 관련된 정보들을 이와 같은 온톨로지 모델로 표현함으로써, 개발자들이 향후 'Draw-Budget' 컴포넌트를 재사용하고자 할 때 'DrawBudget' 컴포넌트의 작성에 대한 근거를 확보함에 따라 'Draw-Budget' 컴포넌트를 다른 어플리케이션에 재사용할 경우, 재사용에 소요되는 시간을 감소시킬 수 있다.

5. 검증

4장에서 재사용 모델이 실제 소프트웨어 개발에 어떻게 적용되는지를 보여 주었다면, 5장에서는 재사용 모델이 소프트웨어 재사용에 있어서의 생산성을 얼마나 향상시킬 수 있을 것인가에 대해 실험을 통해 그 가능성을 보여주고 있다.

5.1 실험설계

본 논문에서 제시한 재사용 모델을 사용함으로써 얻을 수 있는 생산성 향상의 정도를 측정해 보기 위해 다음과 같이 설계된 실험을 진행하였다.

- 실험대상: 29명의 대학원생으로, 대상자들은 실제 소프트웨어 관련 업계의 엔지니어들로 평균 5.8년 정도의 경력을 가지고 있다.
- 실험결과 분석의 단위: 4-5명의 학생으로 구성된 프로젝트 팀
- 실험변수: 재사용 모델
  - √ 팀A, B, C - 코드 기반의 재사용 모델
  - √ 팀 D, E, F - 본 논문에서 제시하는 재사용 모델 평가항목
  - √ 노력비용(Man-Hours): 새로운 서비스를 시스템에 추가하는데 소요되는 노력의 정도 측정을 위한 평가항목
  - √ 재사용된 코드 라인수(LOC): 생산성 측정을 위해 필요한 항목
  - √ 에러 갯수: 개발 작업 자체의 품질을 측정하기 위한 항목
- 실험대상 시스템: 다양한 품목에 대해 공개입찰을 통해 회사의 물자를 조달하는 구매업무 시스템으로 Java와 J2EE 아키텍처를 사용하여 구현되었다.
- 실험 시나리오: 도메인 영역에 속하는 비즈니스 로직 계층과 데이타 계층에 속하는 컴포넌트들의 기본적인 기능들은 이미 개발되어 있는 상태이다. 고객으로부터 새로운 "입찰참여" 서비스를 시스템에 추가해 줄 것을 요청 받았다. 각각의 실험 프로젝트 팀은 고객이 요청한 새로운 서비스를 기존의 컴포넌트를 재사용하여 구축하는 것이 본 실험에서의 임무이다.

기존에 개발된 소프트웨어 컴포넌트들과 관련 소프트웨어 산출물들은 아직 MPC 구조에 따라 패킹되어 있지 않은 상태였다. 두 번째 실험 그룹인 팀 D, E, F의 실험 진행을 위해 시스템 상에 산재되어 있던 소프트웨어 산출물들을 서비스별로 정리하는 작업이 선행되었다. 앞서 밝힌 바와 같이, 본 실험에서도 서비스의 단위는 유스케이스 명세서 상의 각각의 이벤트 흐름으로 한정하였다. 이러한 서비스 기반의 패킹 작업은 어플리케이션 영역에 대해서 이뤄졌으며 도메인 엔지니어링 영역에 속한 산출물들을 MPC로 패킹하는 작업에서는 '회사', '사용자', '품목' 과 같은 주요 도메인 개념을 기준으로 각각의 MPC를 식별하였다. 기개발된 소프트웨어 산출물을 MPC 구조에 따라 패킹하는 작업과 이를 다시 온톨로지 모델로 전환하는 작업에 소요된 소요비용은 17 Man-Hour였다. 이 비용은 팀 D, E, F의 초기투자 비용으로 간주하여 재사용 모델의 생산성 비교시 고려하였다.

5.2 실험결과

팀 A, B, C에게는 기존 컴포넌트의 소스 코드와 여기저기 산재해 있는 관련 소프트웨어 산출물들이 주어졌고, 팀 D, E, F에게는 팀 A, B, C에 주어진 재사용 자료와 각각의 관련 산출물 관련 정보들을 MPC 구조에 따라 명세한 후, 압축된 .ras파일 형태로 주어졌다. 이러한 조건에서 동일한 서비스인 '입찰참여' 기능을 시스템에 구현하도록 한 후 측정한 실험결과값은 표 2와 같다. 본 논문에서 제시하는 재사용 모델의 목적은 이미 존재하는 소프트웨어 컴포넌트를 재사용하는데 소요되는 노력을 감소시키는 데 있다. 이러한 점에서 표 2에서 보여주는 결과는 매우 고무적이라고 할 수 있다.

단순히 소스코드만을 활용하여 필요한 관련 산출물들을 개발자들이 일일이 찾아가며 기존 컴포넌트를 재사용한 팀 A, B, C의 경우 67.9 Man-Hour의 노력이 소요된 데 반해, 컴포넌트와 관련된 모든 정보를 잘 정리된 구조로 제공하고 있는 MPC기반의 컴포넌트 명세서를 활용한 팀 D, E, F의 경우 23.9 Man-Hour만이 소요되었다. 이는 팀 A, B, C의 67.9 Man-Hour와 비교하여 약 64.8%의 노력비용이 감소된 것이다. 팀 D, E, F의 초기투자 노력비용인 17 Man-Hour를 고려한다 하더라도 상당한 비용이 감소되었다는 것을 확인할 수 있다. 원래의 목적이었던 재사용에 소요되는 노력 감소 이

표 2 실험결과

실험변수 평가항목	코드기반의 재사용 모델	MPC 기반의 재사용 모델
노력비용(M-H)	67.9	23.9
에러 개수	54.3	13.3
Newly developed LOC	1210	853
Reused LOC	608	1831
Increased Productivity	7.06%	158.03%

외에도, MPC를 사용한 컴포넌트의 정확한 명세와 이를 바탕으로 한 효율적인 컴포넌트의 검색으로 인해, 신규 작성 코드 대비 재사용 코드의 비율 역시, 코드 위주의 재사용시의 50.2%에서 214%로 4배 가량 증가하는 결과를 보이고 있다.

소프트웨어 개발의 생산성 측정을 위한 모델로는 Gaffney와 Durek의 Simple cost-of-development model[15]을 채택하였는데, 이 모델에 의하면 팀 D, E, F의 재사용으로 인한 생산성 향상 정도가 팀 A, B, C의 19배에 달하고 있다. 마지막으로 측정된 평가항목 중 하나인 개발시 발견된 에러 개수 역시 팀 D, E, F의 에러는 13.3개로 팀 A, B, C의 54.3개에 비해 현저히 감소된 숫자이다. 에러 감소의 이유는 두 가지 요인으로 분석되는데, 첫 번째는 MPC를 사용한 명확한 컴포넌트의 명세로 인한 개발자들의 컴포넌트 이해도 증가이고, 두 번째는 테스트를 거친 검증된 기존 코드의 재사용 비중이 높아짐에 따라 그만큼 소프트웨어 자체의 품질 향상이 있었던 것으로 볼 수 있다.

실험을 통해 본 논문에서 제시하는 재사용 모델의 여러 가지 장점을 발견할 수 있었으나, 실험에 사용된 MPC의 개수는 22개로 휘쳐 그래프의 정보를 바탕으로 하는 유사도 측정을 통한 컴포넌트 검색의 효율성을 점검하기에는 어려움이 있었다. 따라서 표 2에서 보이는 향상된 부분들은 본 논문에서 제시하는 재사용 모델의 구성요소 중에서도 MPC기반의 소프트웨어 명세로 인해 얻을 수 있는 장점으로 해석할 수 있다.

6. 결 론

본 논문에서 제시한 새로운 재사용 모델은 다음과 같은 세 가지 요구사항을 만족시키기 위해 개발되었다.

- 이미 존재하는 소프트웨어 자산을 활용하는데 있어서, 정보부족으로 인한 중복개발로 전반적인 생산성이 저하되는 것을 미연에 방지한다.
- 재사용 모델의 실 프로젝트 적용 시 초기 투자 노력 비용을 최소화한다.
- 컴포넌트의 규모에 대한 논쟁으로 인한 시간소모를 줄이기 위해, 타당한 소프트웨어 재사용의 단위를 제시한다.

이와 같은 요구사항을 만족시키기 위해, 컴포넌트에 대한 다각적인 정보를 MPC라는 RAS의 확장된 구조의 사용을 통해 개발자들에게 제공하며, 서비스를 하나의 MPC에 수록하는 소프트웨어 자산들의 단위로 정의하고 있는 새로운 재사용 모델을 제시하였다. 제시한 모델의 실효성을 보이기 위해 실제적인 사례연구를 수행하였으며, 기존의 코드 위주의 재사용 모델과 비교하여 어떤 이점을 줄 수 있는지에 대한 평가를 계획된 실험 결과로부터 유추할 수 있었다. 이러한 일련의 검증 작업을 통해 우리는 새로운 재사용 모델이 다음과 같은 이점을 줄 수 있다고 분석한다. 아래의 이점들 중에서도, 첫 번째 이점이 실제로 나머지 다른 이점들을 파생시키는 핵심적인 이점이라고 할 수 있다.

- 기개발된 컴포넌트를 재사용하기 위해 필요로 하는 컴포넌트의 유무를 검색하고, 검색된 컴포넌트를 파악하는데 효과적인 방법을 제시함으로써, 전반적인 재사용에 소요되는 노력을 감소시킬 수 있다. 또한, 기존의 제품 라인 공학(Product Line Engineering)과 같은 접근방법과 비교하여, 비교적 적은 초기 노력비용이 투자된다고 볼 수 있다.
- 검증된 소프트웨어 컴포넌트의 재사용을 활성화함으로써, 소프트웨어 개발시 발생하는 에러의 개수를 줄일 수 있다. 이러한 에러 개수의 감소는 소프트웨어 품질 향상으로 연계될 수 있다.
- 결과적으로 신규코드 대비 재사용 코드의 비율이 증가함에 따라 소프트웨어 개발의 생산성 향상을 기대할 수 있다.

결론에서 기술하고 있는 바와 같이 MPC와 서비스 기반의 재사용 모델에 관한 연구의 초기성과는 고무적이라고 볼 수 있다. 그러나, 앞으로 대규모 컴포넌트를 대상으로 한 적용이 이루어지기 위해서는 MPC 등록자의 역할을 자동화 할 수 있는 컴포넌트 검색 엔진 등을 포함하는 자동화된 프레임 워크의 개발이 뒤따라야 한다. 이러한 점들은 아직까지는 해결하지 못한 부분으로, 앞으로 지속적인 연구를 통해 해결해 나갈 계획이다.

참 고 문 헌

[1] B.W. Boehm, M. H. Pendo, A. B. Pyster, E.D. stuckle and R. D. William, "An Environment for Improving Software Productivity," IEEE Computer, June 1984.

[2] Paul C. Clements, "From Subroutines to Subsystems: Component-Based Software Development," The American Programmer, Vol.8, No.11, November 1995.

[3] R. K. Keller and R. Schauer, "Design components: towards software composition at the design level," Software Engineering, Proceedings of the 1998 (20th) International Conference, pp.302-311, April 1998.

[4] OMG, Reusable Asset Specification, Available Specification Version 2.2 formal/05-11-02.

[5] Thomas Erl, Service-Oriented Architecture (SOA): Concept, Technology, and Design, Prentice Hall, 2005.

[6] S. Dakhli and C. Toffolon, "Software Artifacts Reuse and Maintenance : An organizational Frame-

work," 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98), p.228, 1998.
[7] R. G. Fichman and C. F. Kemerer, Incentive Compatibility and Systematic Software Reuse, Journal of Systems and Software, Vol.57, No.1, 2001.
[8] P.Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, Documenting Software Architectures, Addison-Wesley, MA, 2003.
[9] S. C. Chu, "From Component-based to Service Oriented Software Architecture for Healthcare," Enterprise Networking and Computing in Healthcare Industry 2005, HEALTHCOM 2005. Proceedings of 7th International Workshop, pp. 96-100, 23-25 June 2005.
[10] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," Keynote for the 4th International Conference on Web Information Systems Engineering (WISE 2003), IEEE CS, December 10-12, 2003.
[11] Erich Gamma, Richard Helm and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.
[12] Asuncio Gomez-Perez, Mariano Fernandez-Lopez and Oscar Corcho, Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web, Springer-Verlag London Ltd., Oct. 2003.
[13] Ora Lassila, Deborah L. and McGuinness, "The role of Frame-Based Representation of the Semantic Web," KSL Tech Report Number KSL-01-02. Submitted for publication, January 2001.
[14] Eduardo Ostertag, James Hendler, Ruben Prieto Diaz and Christine Braun, "Computing Similarity in a Reuse Library System: An AI-Based Approach," ACM Transaction on Software Engineering and Methodology, Vol.1, No.3, pp. 205-228, July 1992.
[15] J. E. Gaffeney and T. A. Durek, Software reuse-key to enhanced productivity: some quantitative models. Information and Software Technology, Volume 31, Issue 5, pp. 258-267, June 1989.

박 수 진
1995년 경북대학교 컴퓨터공학과(공학사) 2000년 서강대학교 정보통신대학원(공학석사). 1995년~1999년 (주)LG-CNS 근무. 2000년~2002년 한국래쇼날 소프트웨어 선임컨설턴트. 2003년~현재 서강대학교 컴퓨터공학과 박사과정. 관심분야는 소프트웨어 재사용, 소프트웨어 개발 프로세스, 요구공학

박 수 용
1986년 서강대학교 전자계산학과 공학사 1988년 플로리다 주립대 전산학 석사 1995년 George Mason University 정보기술학박사, 연구 조교수. 1996년~1998년 TRW Senior Software Engineer 1998년~현재 서강대학교 컴퓨터공학과 교수. 관심분야는 요구공학, Adaptable Component, Web Service