# 재사용을 위한 컴포넌트 분류체계와 UML을 이용한 컴포넌트 분류체계 표기법에 관한 연구

박진구, 김강태, 이경환
중앙대학교 컴퓨터공학과 소프트웨어공학 연구실

# A Study of Component Categorization for Reuse and Representation with UML

JinGoo Park, GangTae Kim, KyungWhan Lee
SE Laboratory, Dept. of Computer Science & Engineering, Chung-Ang University

**요 약**

컴포넌트 시장이 활성화됨에 따라 컴포넌트의 수는 기하급수적으로 증가하고 있으며, 이러한 컴포넌트들을 효과적으로 재사용하기 위해서는 컴포넌트 분류체계가 잘 정의되어질 필요성이 있다. 현재 컴포넌트 유통시장의 현황을 살펴보니, 대체적으로 구현과 관련된 실행가능모듈로서의 컴포넌트들이 언어, 기능성, 플랫폼정도로 분류가 되어 사용되어지고 있었다. 앞으로 다양한 컴포넌트들이 많이 개발되어질 것이며, 이들을 위한 체계적인 분류방법이 필요하다고 생각되어, 본 논문에서는 효과적인 컴포넌트 재사용을 위한 컴포넌트 분류체계를 제시하였다. 또, 이를 컴포넌트 모델링에 적용하기 위해 제시한 분류체계를 UML의 스테레오타입으로 정의하여 시각화하였다.

## 1. 서론

소프트웨어 개발에 있어서 가장 큰 관심은 소프트웨어 개발 생산성을 높이는 것이다. 소프트웨어 개발 생산성을 높이기 위해서는 소프트웨어를 이루고 있는 여러 가지 구성단위들을 재사용하는 방법이 가장 효과적이라고 판단되었다. 재사용단위로서 컴포넌트의 개념이 도입되면서, 소프트웨어 개발 생산성은 비약적으로 높아지게 되었다.

컴포넌트 시장이 활성화됨에 따라 컴포넌트의 수는 기하급수적으로 늘어나고 있으며, 효과적인 컴포넌트 재사용을 위해서는 체계적인 컴포넌트 분류체계가 꼭 필요하다. 현재 컴포넌트 유통시장의 현황을 살펴보면, 구현언어나 플랫폼, 기능성에 치중한 비교적 간단한 분류방법을 사용하여 컴포넌트를 공급하고 있으며, 재사용가능한 컴포넌트들은 대체적으로 개발 라이프사이클중 구현시 사용될 수 있는 것들이 대다수이다.

컴포넌트를 재사용하려는 사람은 자신이 찾고자 하는 컴포넌트를 체계적인 방법으로 쉽게 찾고 알아볼 수 있어야 하며, 마찬가지로, 재사용가능한 컴포넌트를 만들어 제공하려는 사람도 자신의 컴포넌트가 어떻게 재사용되어질 수 있는지를 체계적인 분류체계에 따라 사용자에게 나타낼 수 있어야 한다. 이를 통해 효과적인 컴포넌트 재사용이 가능하며, 이는 소프트웨어 생산성 향상에 직결될 것이다.

본 논문에서는, 재사용 가능한 컴포넌트를 만들어 제공하고자하는 사람과 그 컴포넌트를 재사용하려는 사람간에 공감할 수 있는 체계적인 컴포넌트 분류체계를 제안하고, 이를 효과적으로 시각화하여 나타낼 수 있도록, UML(Unified Modeling Language)의 스테레오타입을 이용한 표기법을 제안한다.

## 2. 기반연구

### 2.1 컴포넌트의 정의

컴포넌트에 대한 다양한 관점들이 존재하며 이에 따른 정의들도 매우 다양하다. 바라보는 관점에 따른 컴포넌트 정의들은 다음과 같다.

- 컴포넌트는 기능적인 시스템을 형성하기위해 인터랙트하는 독립적인 생산, 획득, 배치의 이진 단위이다. - C.Szyperski
- 컴포넌트는 자질구레하지 않으며(non-trivial), 거의 독립적이고 대체가 가능한 시스템 부품이다. 이 컴포넌트는 잘 정의되어진 아키텍처의 컨택스트내에서 명확한 기능을 만족시켜주며, 인터페이스 집합에 대한 실제 구현물이라 할 수 있다. - Krutchen[4]
- 컴포넌트는 독립적으로 개발되어지고 배포되어질 수 있는 응집된 소프트웨어 패키지이다. 컴포넌트가 제공하는 서비스와 다른 컴포넌트들로부터 요구하는 서비스에 대한 명시적이고 잘 정의된 인터페이스들을 갖는다. 또한, 이러한 컴포넌트들의 프로퍼티에 대한 커스터마이징이 가능하다. - D'Souza[1]

2.2 컴포넌트의 특성

일반적으로 알려져 있는 컴포넌트의 특성으로 다음과 같은 항목이 포함된다.

| 특성 | 설명 |
|---|---|
| Encapsulated | 좋은 컴포넌트는 예측가능하며, 잘 정의되어진 인터페이스를 갖는 블랙박스이다. |
| Nearly independent | 컴포넌트는 반드시 다른 컴포넌트들에 대해 낮은 결합도를 가져야 한다. |
| Highly cohesive | 컴포넌트의 목적이 명확하고, 정확한 초점을 가지려면, 강한 응집도를 유지해야 한다. |
| Trusted and marketable | 기존의 컴포넌트는 반드시 어떤 형태로든 구현되어져 가치를 측정할 수 있어야하며, 문서화되어져야 한다. |
| Reusable | 컴포넌트가 설계되고, 구축되고, 판매되어진 후에는, 모든 사용자들이 그 컴포넌트가 제공하는 서비스를 수행하기 위해 그 컴포넌트를 구입, 사용할 수 있다. |
| Replaceable | 컴포넌트는 최소한 같은 인터페이스들을 통해 같은 서비스들을 제공하는 새로운 컴포넌트들로 쉽게 대체되어질 수 있어야 한다. |
| Executable | 컴포넌트는 모든 사용자들에 의해 실행가능해야 한다. |
| Distributable | 컴포넌트들은 표준 컴포넌트 인터페이스나 미들웨어 서비스를 통해 분산환경에서 실행가능해야 한다. |
| Scalable | 하나의 컴포넌트는 다른 여러 서버들 상에서 실행될 수 있도록 설정가능해야 한다. |
| Interoperability | 컴포넌트 인터페이스와 미들웨어 서비스들의 표준에 관련되어 있는 한, 반드시 모든 플랫폼에서 그 컴포넌트의 서비스들을 요청할 수 있어야 한다. |
| Self-describing | 컴포넌트는 반드시 그 컴포넌트의 공용 인터페이스와 커스터마이징이 가능한 프로퍼티들, 그리고 그 컴포넌트가 생성하는 이벤트들을 묘사할 수 있어야 한다. |

2.3 기존의 컴포넌트 분류방법들

John Cheesman[2]은 컴포넌트를 다음 세가지 perspective로 나누어 분류하였다.

| Perspective | explanation | element |
|---|---|---|
| Packaging perspective | 패키징, 분산, 배포단위로서의 컴포넌트 | Files, Documents, Source code files, Class libraries, Templates, tables, dll's |
| Service perspective | 서비스의 제공자로서의 컴포넌트 | Database services, Operating system services, System libraries, Individual API functions, COM classes |
| Integrity perspective | 통합이나 캡슐화의 경계로서의 컴포넌트 | Databases, Operating systems, Frameworks, ActiveX controls, Some COM classes, Java Applets |

C.Frye[3]는 논리적 견해와 물리적 견해로 나눈 분류방법을 사용하였으며, 이에 따라 컴포넌트 타입이 결정되어진다고 하였다. 논리적 견해는 컴포넌트들을 비즈니스 영역내의 실세계 개념들을 모델링하는 수단으로서 보는 것이며, 여기에는 Business components가 포함된다. 물리적 견해는 컴포넌트들을 그러한 실세계 비즈니스 개념들을 구현한 소프트웨어, 혹은 어플리케이션 빌딩 블록으로서의 독립된 조각으로 보는 것이다. 여기에는 Partitioned Business Components와 Engineering Components가 포함된다.

W. kozaczynski[4]는 하나의 컴포넌트는 그 컴포넌트가 가진 프로퍼티나 컨택스트에따라 다양한 뷰들을 가지게 된다고 하였으며, 컴포넌트를 크게 Design view, Implementation view, Deployment view인 세가지 뷰의 관점에서 분류하였다.

## 3. 재사용기반 컴포넌트 분류체계

3.1 재사용기반 컴포넌트 분류체계의 제안

컴포넌트를 재사용하기위해서는 컴포넌트를 재사용하려는 사람과 재사용가능한 컴포넌트를 제공하려는 사람간에 공감할 수 있는 컴포넌트 분류체계가 존재해야하며, 이를 통해 효과적이고, 쉽고, 직관적인 컴포넌트 재사용이 이루어질 수 있다. 이러한 컴포넌트 분류체계 확립을 위해서, 일반적인 컴포넌트 특성들과 여러 컴포넌트 정의들을 기반으로 분류기준을 정의하였다. 본 논문에서 정의한 분류기준은 다음과 같다.

1. Purpose

컴포넌트는 목적에 따라 실세계문제를 지원하는지 혹은 어플리케이션 개발의 기술적인 면을 지원하는지에 따라 분류될 수 있다. 실세계 문제를 지원하는 컴포넌트를 Business 컴포넌트라 정의한다, 이는 일반적인 비즈니스 영역내의 개념들과 문제를 표현하며, 주로 이 영역내의 사용자들이 원하는 문제 해결을 위해 사용한다. 실제적인 어플리케이션 개발의 기술적인 면을 지원하는 컴포넌트를 Technical 컴포넌트라 정의하며, 이는 특정 어플리케이션들의 구축을 지원하기 위해 사용되어지는 컴포넌트이다. 이 Technical 컴포넌트는 기술적인 목적을 띄며, 소프트웨어 엔지니어링 요구사항들에 초점을 맞추고 있으며, 특정한 실세계 문제에 종속되지 않는다. 주로 개발자에 의해 사용된다.

2. Scope

컴포넌트는 범위에 따라 크게 문제를 기술한다는 측면과 구현을 지원한다는 측면 두 가지로 생각할 수 있으며, 전자를 Specification 컴포넌트, 후자를 Implementation 컴포넌트로 정의한다. Specification 컴포넌트는 비즈니스 문제나 기술적인 문제의 단어와 구조를 정의한 기술서라 할 수 있으며, 다이어그램과 문서로 기술된다. Specification 컴포넌트는 세부적으로 컴포넌트 타입, 요구사항분석, 설계패턴, 컴포넌트 모델 등이 포함된다. Implementation 컴포넌트는 특정 구현기술이 사용되어져 완성된 컴포넌트의 집합이며, 여기에는 소스코드나 실행가능한 코드, 컨트롤, 어플리케이션 프레임워크, 어플리케이션 객체 등이 포함된다.

3. Granularity

컴포넌트는 다양한 크기와 복잡도를 갖는다. 이를 판단할 수 있는 기준으로서 granularity를 정의하며, fine-granularity와 coarse-granularity로 나뉜다. 전자는 규모가 작으며, 덜 복잡하며, 한정된 기능을 제공하는 반면, 후자는 규모가 크며, 더 복잡하고, 확장된 기능을 제공한다.

4. Interface

컴포넌트는 다른 컴포넌트에게 제공하는 서비스와 제공받는 서비스를 위해 인터페이스를 갖는다. 이 인터페이스를 통해서만 컴포넌트는 다른 컴포넌트와 커뮤니케이션이 가능하다.

컴포넌트 인터페이스는 어플리케이션 인터페이스와 플랫폼 인터페이스로 나뉜다. 어플리케이션 인터페이스는 플랫폼에 상관없이, 컴포넌트들이나 어플리케이션 개체들과의 인터랙션이 명세된 것이다. 플랫폼 인터페이스는 그 컴포넌트가 실행되는 플랫폼과의 인터랙션을 정의하며, 여기에는 OS call, 하드웨어기술, 커뮤니케이션 서브시스템들에 대해 명세된다.

5. Context adoption level

컴포넌트는 컨택스트의 적용 레벨에 따라 Horizontal 컴포넌트와 Vertical 컴포넌트로 분류한다. 어떤 컴포넌트가 둘 이상의 컨택스트내의 어플리케이션들에 적용될 수 있다면 이는 Horizontal 컴포넌트이다. 만약, 단 하나의 특정 컨택스트에 한해 사용될 수 있는 컴포넌트라면, 이는 Vertical 컴포넌트이다.

6. Deliverable Form

컴포넌트의 배포형태에 따른 분류로서, 크게 Design component, Executables, Source code로 나눌 수 있다. Design 컴포넌트는 설계원칙 혹은 아이디어가 될 수 있으며, 객체지향 프레임워크를 구축하는데에 여러 디자인 패턴들이 디자인 빌딩 블록으로서 사용되어질 수 있다. Executables는 정적라이브러리, 동적 링크라이브러리(DLL), 실행가능한 어플리케이션 코드조각 형태로 존재하며, 현재 거의 모든 컴포넌트 관련 벤더들은 실행가능한 형태로 컴포넌트를 제공하며, 설계코드나 구현코드는 재산가치를 지니는 프로덕트로서 보유하고 있는 경향이 많다. Source code는 다른 프로젝트에서 재사용되어지거나 테스트되어지는데 사용되어질 수 있고, 이를 분석하여 더욱 향상된 컴포넌트를 만들어낼 수 있다. 보통은, 같은 영역이나, 같은 조직내에서 다른 어플리케이션들에 사용되어질 수 있는 코드 라이브러리로서 사용되어진다.

7. Reuse level

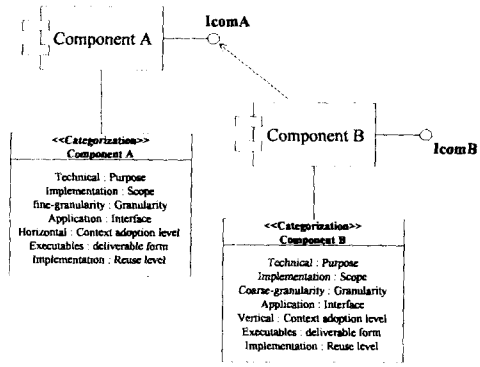
이는 소프트웨어 개발 라이프사이클(계획, 요구사항분석, 설계, 구현, 통합 및 테스트, 운영 및 유지보수)단계별로 재사용될 수 있는 단계에 따라 컴포넌트를 분류한 것이다. 예를들어, 코드조각은 구현시에, DLL은 소프트웨어 실행시에, 그리고 정적 라이브러리는 어플리케이션 통합시에 재사용될 수 있다.

### 3.2 UML을 이용한 컴포넌트 분류체계의 표현방법

현재 UML[5]에서 표준으로 정의되어진 컴포넌트 타입에 대한 스테레오 타입은 문서를 나타내는 <<documentation>>, 노드에서 실행가능한 컴포넌트<<executables>>, 소스코드나 데이터를 수록하고 있는 문서를 나타내는 <<file>>, 정적 혹은 동적 객체 라이브러리를 나타내는 <<library>>, 데이터베이스 테이블을 나타내는 <<table>> 다섯가지이다. 이에, UML을 이용한 컴포넌트 모델링시에 앞절에서 제시되었던 컴포넌트 분류체계를 효과적으로 시각화하기 위해, 분류체계를 위한 스테레오타입 <<categorization>>을 표 1과 같이 정의한다. 그리고, 이를 UML을 이용한 컴포넌트 모델링에 적용한 예는 그림 1과 같다.

| 분류기준 | 세부 분류 사항 |
|---|---|
| Purpose | Business, Technical |
| Scope | Specification, Implementation |
| Granularity | fine-granularity, coarse-granularity |
| Interface | Application, Platform |
| Context adoption level | Horizontal, Platform |
| Deliverable Form | Design component, Executables, Source code |
| Reuse level | Planning, Requirement analysis, Design, Implementation, Integration and Testing, Maintenance |

[표1] 스테레오타입 <<categorization>>의 정의




[그림1] 분류체계를 컴포넌트 모델링에 적용한 모습

## 4. 결론 및 향후 연구과제

본 논문에서는 재사용을 위한 컴포넌트 분류체계를 제시하고, UML을 이용한 표현방법에 대해 연구하였다. 이미 알려진 여러 컴포넌트 정의들을 참조하고, 일반적인 컴포넌트 특성들을 이해함으로써, 앞으로 크게 활성화되어질 컴포넌트 시장에 대비해, 수많은 사용가능한 컴포넌트들을 효과적이고 쉽게 재사용할 수 있는 컴포넌트 분류체계를 제시하였고, 이를 컴포넌트 모델링에 적용하기위해, UML의 스테레오타입을 사용한 시각화방법을 제시하였다.

향후 본 논문에서는 제시된 컴포넌트 분류체계를 특정 어플리케이션 개발에 실제 적용하여 재사용 효과를 알아보고, 평가해본다. 또, 이를 기반으로 하는 CBD 방법론으로의 확장 및 적용가능성을 연구할 것이다.

## 5. 참고문헌

[1]Desmond F. D'Souza, *Objects, components and frameworks with UML*. Addison-Wesley, Reading, 1998

[2]J.Cheesman.,*What is a Component?*, Sterling Software, 5 February 1998

[3]C.Frye.,*Understanding Components*, Anderson Consulting, 1 May 1998

[4]W.Kozaczynski, *Composite Nature of Component*,1999

[5]Grady Booch, James Rambaugh, Ivar Jacobson; *The Unified Modeling Language User Guide*, Addison Wesley, 1999