6 View기반 컴포넌트 분류 및 명세 기법
(Techniques for Classifying and Specificatying Components based on Six Views)

조 은 숙 † 이 종 국 †† 김 수 동 †††
(Eun Sook Cho) (Jong Kook Lee) (Soo Dong Kim)

요 약 컴포넌트 기반의 재사용 기술이 소개되면서 소프트웨어 컴포넌트의 유통이 인터넷을 통한 온라인 기반의 유통 형태로 변하게 되었다. 이를 위해서는 유통 모델이 필요하며, 유통 시스템의 구축이 이루어져야 한다. 더욱이 유통 시스템이 효율적으로 운영되기 위해서는 컴포넌트들을 효율적으로 관리, 검색하기 위한 분류 체계가 마련되어야 한다.

본 논문은 이러한 유통 시스템 구축에 필요한 컴포넌트 분류 체계를 6가지 관점을 기반으로 한 컴포넌트 분류체계를 제시하고 BNF 표기법을 이용하여 명세한다. 제시된 분류체계의 효율성을 검증하고 기존의 분류체계들과 비교하기 위해 개발된 컴포넌트들을 적용하여 적중율과 정확도를 측정하여 실험 및 평가한다. 본 논문에서 제시한 기법이 기존의 분류기법에 비해서 여러각도에서 분류하기 때문에 컴포넌트의 검색이나 등록이 효율적으로 이루어질 수 있도록 한다.

키워드 : 컴포넌트 기반 재사용, 유통 모델, 유통 시스템, 컴포넌트 분류 체계

Abstract As component-based reuse technology is being introduced, software components are more likely distributed on Internet. In order to promote Internet based on-line distribution, distribution model of components should be defined and a distribution system should also be constructed. Furthermore, to run the component distribution system effectively, a logical and practical schema for classifying components should be defined and standardized.

In this paper, we propose a classification schema using BNF and representation standards of components using six different views. To verify the effectiveness of proposed classification and compare to other classification methods, we assess proposed classification by measuring hit ratio and correctness. By using the proposed methods that support six views on components, the search for right components and registering new components can be done more effectively.

Key words : Component-based Reuse, Distribution Model, Distribution System, Component Classification Schema

1. 서 론

1970년대 소프트웨어 위기 문제가 대두되면서 소프트웨어의 재사용에 대한 연구는 현재까지 지속적으로 진행되어 왔으나 하드웨어 재사용 기술의 빠른 발전 속도에 비해 소프트웨어 재사용의 발전 속도는 그다지 빠르게 진행되지 못해왔다. 또한 소프트웨어 재사용 기술이 프로시저(Procedure) 재사용에서부터 출발하여 객체지향으로 오면서 클래스 재사용에 이르기까지 여러 가지 재사용 기술들이 소개되어 왔으나 실제 소프트웨어 개발에 있어서 재사용의 효과를 충분히 거두지 못한 실정이다. 이러한 문제를 새롭게 극복하기 위해 컴포넌트 기술이 새로운 재사용 기술로 소개되기 시작하였으며 1990년대 중반 이후에 접어들면서 컴포넌트 개발과 컴포넌트 기반의 소프트웨어 개발 패러다임이 활발히 진행되기 시작했다.

현재 전세계적으로 컴포넌트에 대한 많은 연구와 개발들이 전세계적으로 활발히 진행되고 있으나 하드웨어의

† 정 회 원 : 동덕여자대학교 정보학부 교수
escho@dongduk.ac.kr
†† 비 회 원 : 숭실대학교 컴퓨터학과
jklee690@selab.soongsil.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2002년 3월 12일
심사완료 : 2002년 5월 23일

재사용 수준에 이르기에는 아직 많이 미흡한 상태이다. 특히 국내 같은 경우는 현재 컴포넌트에 대한 개념을 정립해 나가고 있는 상황이며 일부 업체들이 컴포넌트 기술 플랫폼을 바탕으로 특정 도메인에 필요한 컴포넌트 개발을 시도하고 있는 실정이다. 그러나 아직 컴포넌트를 효율적으로 개발하기 위한 방법론이나 컴포넌트 품질 보증 기법, 컴포넌트 시험 기법, 다양한 분야의 컴포넌트들에 대한 표준 정립 등과 같은 많은 기술들이 아직 미성숙한 상태라 볼 수 있다. 특히, 개발된 컴포넌트들을 효율적으로 관리하고 운영 및 유통하기 위한 컴포넌트 분류체계에 대한 연구는 거의 이루어져 있지 않다.

물론 현재 외국 사례를 보면 Component Source[1]나 Flashline[2]과 같이 컴포넌트 유통사이트를 통해 컴포넌트의 유통이 이루어지고 있으나 등록된 컴포넌트들이 사용자 인터페이스 용도의 컴포넌트들이 많이 있으며 또한 컴포넌트라기 보다는 어플리케이션 제품들이 등록되어 있는 경우들도 많다. 따라서 컴포넌트를 검색해 보면 재사용하고자 하는 컴포넌트가 아니라 실제 소프트웨어가 해당 분야의 컴포넌트로 등록되어 유통되고 있는 경우도 볼 수 있다. 또한 유통 사이트들의 컴포넌트에 대한 분류체계를 보면 분류체계가 단순히 기능적 관점에서만 분류되어 있는 실정이다. 따라서 많은 회사들이 컴포넌트들을 개발하여도 이를 인터넷과 같이 네트워크 상에서 유통하기에는 아직 미약한 실정이며, 또한 컴포넌트 레포지토리에 등록할 경우에 있어서도 저장체계가 체계적으로 마련되어 있지 않다.

따라서 본 연구에서는 이에 대한 효율적인 등록, 검색, 그리고 유통하기 위한 분류체계를 기존의 단순한 기능적인 중심 분류체계에서 벗어나 6가지 관점에서 분류한 컴포넌트 분류체계를 정의하여 제시하고자 한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구로 기존 컴포넌트 분류체계들의 형태와 기존 분류체계들이 가지고 있는 한계점을 살펴본다. 3장에서는 본 논문에서 제시하는 6가지 컴포넌트 분류 기준에 대해서 설명한다. 4장에서는 본 논문에서 제시하는 컴포넌트 분류 체계에 대한 코드 체계를 명세한다. 5장에서 6가지 관점을 기반으로 한 컴포넌트 분류체계를 적용한 사례 연구를 설명하고, 마지막으로 6장에서 결론 및 향후 연구과제를 제시한다.

2. 관련 연구

2.1 컴포넌트소스(Componentsource)의 분류체계

이 시스템은 컴포넌트 소프트웨어만을 대상으로 확보하여 성장한 업체로 카드결재를 통해 컴포넌트를 구매 후 다운로드 받아 즉시 사용할 수 있는 유통모델을 갖고 있다[1]. 이 시스템의 특징은 컴포넌트 시험 테스트를 실시하여 자체내의 엄격한 테스트를 거쳐 상품을 등록한다. 평가판 다운로드 서비스와 글로벌 언어와 다양한 통화단위(Currency)를 제공하며 업그레이드 센터를 운영하여 제작사 및 상품명 별로 컴포넌트 검색이 가능하도록 하였다.

이 시스템의 분류체계가 가지고 있는 장점은 카테고리 명을 보고도 개발자가 쉽게 찾을 수 있도록 분류체계를 세분화하였다는 것이다. 단점은 하위 분류 체계 없이 바로 소분류로 구성하여 체계적으로 컴포넌트 분류가 되어있지 않고, 향후 컴포넌트 유형이 많아지면 컴포넌트 분류체계가 너무 산만하고 복잡하다.

2.2 플래쉬라인(Flashline)의 분류체계

플래쉬라인은 비즈니스를 위한 소프트웨어 시스템을 보다 빠르게 개발할 수 있도록 지원하는 소프트웨어 컴포넌트 제품, 서비스, 그리고 여러 자원들을 제공하는 회사로서, 웹을 통해 온라인으로 다양한 소프트웨어 컴포넌트들을 유통시키고 있다. 현재 JavaBeans[3], EJB[4], 그리고 COM[5] 컴포넌트를 온라인으로 판매 유통하고 있을 뿐만 아니라 품질 보증, 오픈 컴포넌트 등록 등과 같이 다양한 자원들을 함께 제공하고 있다[2] .

플래쉬라인에서는 컴포넌트 분류체계를 크게 대분류, 중분류, 소분류 체계로 구성되어 있다. 우선 대분류에서는 컴포넌트 아키텍쳐(플랫폼)를 기반으로 Java와 COM으로 분류하고 있다. 다음으로 중분류 체계로서 기술(Technology), 인터넷/웹(Internet/WWW), 통신, 정보관리, 사용자 인터페이스, 교육, 개발도구, 유틸리티, IDE Extensions에 따른 분류체계를 정의하고 있다. 그리고 각각의 중분류 체계별로 소분류 체계들을 정의하고 있다.

이 분류체계가 가지는 장점으로는 우선 컴포넌트를 대분류, 중분류, 소분류로 나누어서 컴포넌트들을 효율적으로 관리하는 분류체계를 지니고 있다는 것이고, 다음으로는 컴포넌트 사용자들로 하여금 컴포넌트들을 한번 돌려볼 수 있도록 데모(Demo) 버전을 제공한다는 것이다. 그러나 이 사이트의 분류체계를 보면 크게 3가지 형태의 분류 체계를 가지고 있는데, 현재의 분류체계만 가지고는 비즈니스 도메인에 대한 분류체계가 미흡하다고 볼 수 있다. 예를 들어, 은행과 관련된 컴포넌트는 현재 이 사이트에서 제공하는 분류체계에 적합한 분야가 기술과 관련된 분류체계밖에 속하지 않는다. 그리고, 중분류체계에 대한 기준이 혼합되어 있다. 즉, 기술분야, 통신 분야, 유틸리티 분야 등과 같이 분류 레벨이

서로 다른 기준이 함께 중분류속에 정의되어 있음으로 인해서 사용자들로 하여금 혼동을 가져올 수 있다. 마지막으로 이 사이트에서 검색된 컴포넌트들에 대한 정보를 보면 컴포넌트의 인터페이스에 대한 정보가 미약하고, 컴포넌트 사용자가 특화(Customization) 할 수 있는 부분에 대한 정보도 제공되고 있지 않다.

2.3 ImagiCom의 분류체계

이 시스템의 분류체계는 크게 3가지 형태로 컴포넌트 분류체계를 제공하고 있다. 컴포넌트 타입, 파일 타입, 그리고 업체별로 분류하고 있다[6].

이 시스템의 분류체계의 장점은 사용자들에게 3가지 관점에서 컴포넌트들을 검색할 수 있도록 제공한다는 것이다. 그러나 단점으로는 크게는 3가지 관점으로 분류를 하였지만 각각의관점으로 들어가 보면 분야별로 세분화되어 있지 않아 특정한 컴포넌트를 검색하길 원하는 사용자에게는 분류체계가 모호하게 생각될 수 있으며, 초보자에게는 광범위한 카테고리가 검색을 어렵게 만든다. 또한, 비즈니스 컴포넌트들에 대해서 영역별로 분류한 분류체계가 제공되지 않고 있다.

2.4 샌프란시스코(Sanfrancisco)의 분류체계

샌프란시스코(SanFrancisco)는 IBM에서 개발한 프레임워크로서 솔루션 제공자(Solution Provider)들이 특화된(Customized) 멀티 플랫폼 시스템들을 구축하는데 있어 복잡도와 비용을 줄일 수 있는 응용 비즈니스 컴포넌트들을 제공한다[7].

샌프란시스코의 분류체계는 그림 1에 표현된 것처럼 대분류로 3개의 통합 계층인 기반 계층(Foundation Layer), 공용 비즈니스 객체 계층(Common Business Objects Layer: CBOs Layer), 핵심 비즈니스 프로세스 계층(Core Business Processes: CBPs Layer) 등으로 분류하고 있다. 그리고 다시 계층별로 중분류체계를 정의한다. 예를 들어 핵심 비즈니스 프로세스 계층의 중분류로는 General Ledger, Warehouse Management, Order Manage, Accounts Payable/Accounts Receivable 등으로 분류하고 있다.

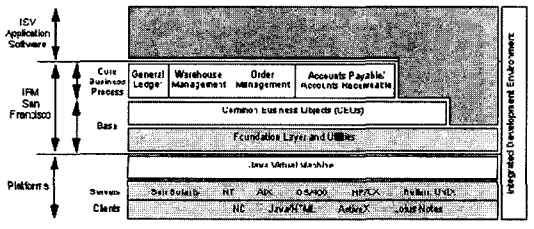

그림 1 샌프란시스코 분류체계

샌프란시스코 분류체계는 컴포넌트의 범용성 차원에서 계층별로 체계적으로 정의하고 있다. 그러나 샌프란시스코의 분류체계는 컴포넌트의 크기로 보면 프레임워크에 해당하는 체계를 지니고 있다. 그리고 이 프레임워크는 하부 서비스에 관련된 부분들과 비즈니스 도메인들 간에 공통된 부분과 특정 비즈니스 도메인에 구체적인 부분들을 계층으로 구별하고 있다. 따라서 이 프레임워크는 범용성의 측면에서만 컴포넌트들을 분류한 아키텍쳐이기 때문에 컴포넌트 검색에 있어서 다양한 경로를 제공하는데 한계점이 있다. 또한 현재 공용 비즈니스 객체 계층에서 구별하고 있는 범주가 금융과 관련된 분야에 초점이 맞춰져 있으며 프레임워크로 제공하고 있기 때문에 어플리케이션 개발자들이 쉽게 접근하기가 어려운 점이 있다.

2.5 UML의 분류 체계

UML은 세 가지로 컴포넌트 종류를 구별하고, 이들 사이의 연결이나 조합에 의해 시스템을 구성하는데 다섯 가지의 스테레오 타입으로 표현할 수 있다[8]. 세 가지로 분류하는 기준은 개발 단계별로 분류한 것이다. 세 가지 분류는 배치 컴포넌트(Deployment Component), 작업 프로덕트 컴포넌트(Work Product Component), 실행 컴포넌트이다. 그리고 이러한 세 가지 분류의 소분류 항목들에 대해서는 스테레오 타입으로 정의하는데 그 종류에는 실행 컴포넌트(Executable Component), 라이브러리 컴포넌트(Library Component), 테이블 컴포넌트(Table Component), 파일 컴포넌트(File Component), 도큐먼트 컴포넌트(Document Component) 등으로 분류한다.

UML에서 정의하고 있는 컴포넌트 분류는 개발 단계별 산출물의 특징을 바탕으로 한 컴포넌트 분류체계이다. 예를 들어 특정 비즈니스 컴포넌트나 사용자 인터페이스 컴포넌트들도 실행 코드 수준의 컴포넌트라면 모두 실행 컴포넌트 분류에 속하게 된다. 따라서 컴포넌트 분류체계가 너무 개략적으로만 분류되어 있다는 한계점이 있다.

2.6 ETRI의 분류체계

ETRI의 분류체계는 크게 기능적인 요소 분류 체계와 비기능적인 요소 분류 체계로 나누어서 분류 체계를 정의하고 있다. 기능적인 요소 분류 체계는 대분류로 도메인 컴포넌트(Domain Component), 공용비지니스 컴포넌트(Common Business Component), 핵심비지니스 컴포넌트(Core Business Component), 기반 비즈니스 컴포넌트(Base Business Component), 아키텍쳐 컴포넌트(Architecture Component) 등으로 분류 하며, 각각의 대분류에 대해서 중분류, 소분류 체계로 확장해서 정의

하고 있다[9].

이 분류체계는 기능적인 측면에서는 체계적으로 컴포넌트 분류체계들을 정의하고 있다. 그러나 또한 다른 분류 체계들과 유사하게 기능중심으로만 컴포넌트 분류체계들을 정의하고 있음으로 인해서 컴포넌트 검색 경로가 다양하지 못하고, 특히 사용자 인터페이스와 관련된 컴포넌트나 시스템 서비스, 유틸리티와 같은 컴포넌트들은 이 분류체계에서는 고려하고 있지 않다.

3. 컴포넌트 분류 체계 기준

본 연구에서는 컴포넌트에 대한 분류체계를 그림 2에 제시된 것처럼 6가지 각도에서 분류하고자 한다. 이는 기존의 분류 체계들이 가지는 단일 관점에서 즉 일반성 또는 기능중심 각도에서만 분류함으로써 서로 다른 유형에 속하는 컴포넌트들이 같은 부류에 속하게 되는 문제점들을 야기할 뿐만 아니라 컴포넌트 사용자들로 하여금 검색된 컴포넌트의 결과가 사용자가 원하지 않는 불필요한 컴포넌트들이 검색되는 경우가 발생하는 문제점들을 해결하기 위함이다.

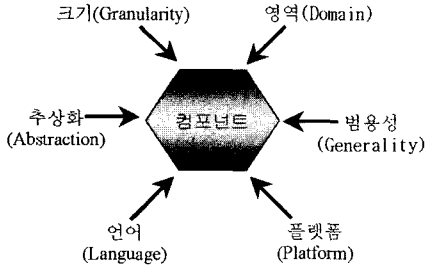

그림 2 컴포넌트 분류를 위한 6가지 각도

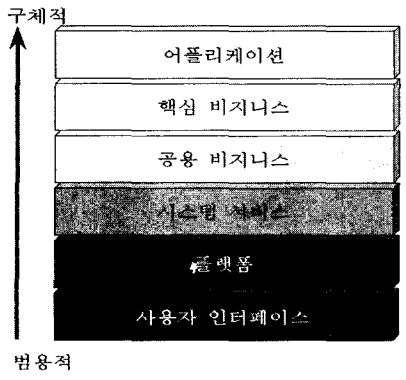

그림 3 범용성에 따른 컴포넌트 분류 체계

3.1 범용성(Generality)에 따른 분류

범용성이라 함은 해당 컴포넌트가 사용되는 범위의 정도를 말하는 것이다. 즉 범용성의 정도가 큰 컴포넌트는 해당 컴포넌트의 사용 범위가 넓다는 것을 의미하고 범용성의 정도가 작은 컴포넌트는 해당 컴포넌트의 사용 범위가 좁다는 것을 의미한다. 본 연구에서는 그림 3에 표현된 것처럼 범용성에 따른 컴포넌트를 크게 6가지로 계층으로 구별하여 규정한다.

3.1.1 사용자 인터페이스 계층

사용자 인터페이스 계층은 범용성에 따른 컴포넌트 분류 체계에서 가장 하위 레벨에 존재하는 계층으로 특정 도메인에 상관없이 여러 다양한 도메인에 사용될 수 있다. 본 논문에서는 사용자 인터페이스 컴포넌트 계층에 대해서 다시 하위 분류체계를 정의하는데 기존의 썬(Sun)에서 개발한 AWT와 스윙(Swing) 패키지, 그리고 기존의 대표적인 컴포넌트 유통 사이트인 컴포넌트 소스와 플래쉬라인과 같은 사이트에서 사용하는 분류체계들을 바탕으로 개발하였다.

기존의 유통사이트들에서의 분류체계에서는 유사한 컴포넌트들이 서로 관련지어서 계층적으로 구성되어 있기 보다는 여러 다양한 컴포넌트들을 한 레벨에서 분류를 한 것을 볼 수 있다. 그러나 이렇게 함으로써 새로운 유형의 컴포넌트들이 생길때마다 분류 체계들의 항목이 방대하게 늘어남으로 인해서 컴포넌트들을 효율적으로 검색, 또는 관리하기가 매우 어렵게 된다. 본 연구에서는 이러한 문제점을 보완하여 보다 관련된 유사 컴포넌트들은 계층적으로 분류하여 분류체계를 구성하여 제시한다. 사용자 인터페이스 계층의 일부 분류 체계가 그림 4에 묘사되어 있다.

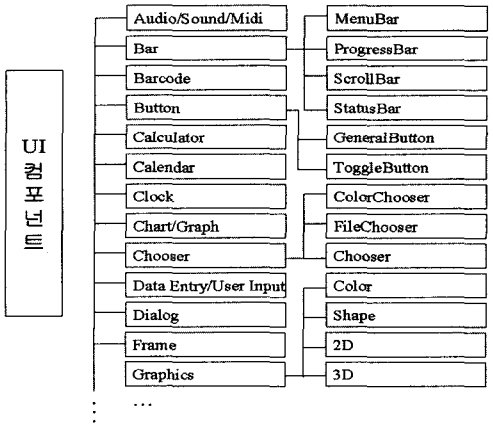

그림 4 사용자 인터페이스 계층의 분류체계

3.1.2 시스템 서비스 계층

시스템 서비스 컴포넌트 계층은 이 분류 체계의 상위 계층인 공용 비즈니스 컴포넌트나 핵심 비즈니스 컴포넌트들이 필요로 하는 하부 시스템서비스 기능들을 제공하는 서비스 컴포넌트와 유틸리티 컴포넌트들을 관리하기 위한 계층이다.

이 시스템 서비스 계층에 소속되는 컴포넌트들은 OMG의 CORBA[10] 아키텍처에서의 서비스 계층에 해당하는 서비스 객체들이나 마이크로소프트(Microsoft)사에서 제안한 .NET 프레임워크에서 제공하는 COM+ 컴포넌트들이 여기에 들어올 수 있다. 시스템 서비스 컴포넌트 계층은 다시 크게 대분류로 서비스 컴포넌트와 유틸리티로 구분되어 있고, 그 밑에 소분류로 여러 세부 분류 항목들로 구성되어 있다(그림 5 참조).

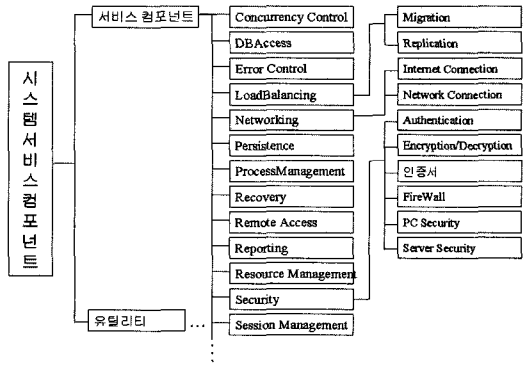

그림 5 시스템 서비스 계층의 분류체계

3.1.3 공용 비즈니스 계층

공용 비즈니스 계층은 비즈니스 컴포넌트들을 포함하는 계층으로서 핵심 비즈니스 컴포넌트 계층과의 차이점은 특정 도메인에 종속되지 않는 컴포넌트들을 가지는 계층이다.

예를 들어, 'Address'나 'Currency'와 같은 컴포넌트들은 특정 도메인에 국한되지 않고 여러 도메인에서 사용되는 컴포넌트이다. 이러한 컴포넌트들은 핵심 비즈니스 컴포넌트 분류계층에 소속되지 않고 공용 비즈니스 컴포넌트 분류 계층에 들어오게 된다.

공용비지니스 컴포넌트 분류 계층은 하위 분류 계층으로 2개의 분류 계층인 비즈니스 데이타 분류 계층과 비즈니스 프로세스 분류 계층으로 정의된다. 비즈니스 데이타 분류 계층은 데이타 중심의 컴포넌트들을 가지는 분류 계층이고, 비즈니스 프로세스 분류 계층은 프로세스 중심의 컴포넌트들을 가지는 분류 계층이다.

3.1.4 핵심 비즈니스 계층

핵심 비즈니스 계층은 특정 도메인에 국한된 컴포넌트들을 관리하는 계층이다. 이 계층은 다시 영역 별로 세부적으로 분류된다. 이에 대한 분류는 영역에 따른 분류 기법에서 분류 체계들을 정의한다.

3.1.5 어플리케이션 계층

어플리케이션 계층은 하위 계층의 컴포넌트들을 기반으로 구성된 소프트웨어나 혹은 단독으로 개발된 소프트웨어들을 표현하는 계층이다. 이 계층은 컴포넌트들을 분류하는 계층이라기 보다는 소프트웨어들을 분류하는 계층으로 볼 수 있는데, 여기에서 고려한 이유는 현재 많은 컴포넌트 유통 시스템들에서 독립적으로 실행되는 프로그램들도 컴포넌트로 분류하여 유통시키고 있기 때문이다. 예를 들어, WS-FTP 프로그램과 같은 단일 프로그램도 컴포넌트 유통시스템에서 유통되고 있다. 이런 경우에 이와 같은 프로그램들은 어플리케이션 계층에 속하게 된다.

3.2 영역(Domain)에 따른 분류

영역에 따른 분류는 이전 절에서 언급한 범용성에 따른 분류에 가운데서 핵심 비즈니스 계층에 대해 다시 영역에 따라 분류하는 것이다. 이처럼 영역에 따른 분류를 하는 이유는 동일 계층에 속하는 컴포넌트일지라도 적용되는 영역은 다를 수 있기 때문이다. 예를 들어, 범용성에 따른 분류에서 공용 비즈니스에 속하는 X라는 컴포넌트가 있다고 하자. 이 컴포넌트는 계층에 대해서는 공용 비즈니스 계층에 속하지만 적용되는 영역은 은

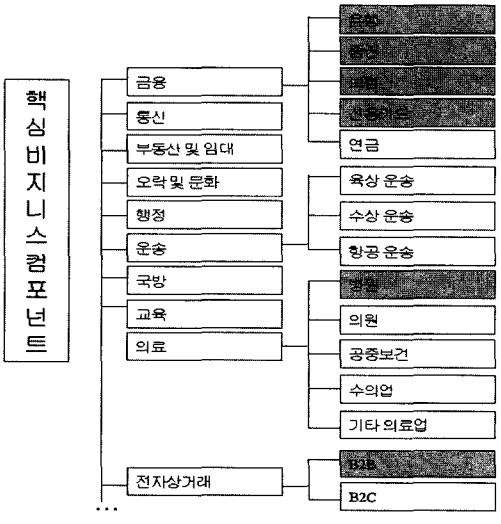

그림 6 영역에 따른 분류체계

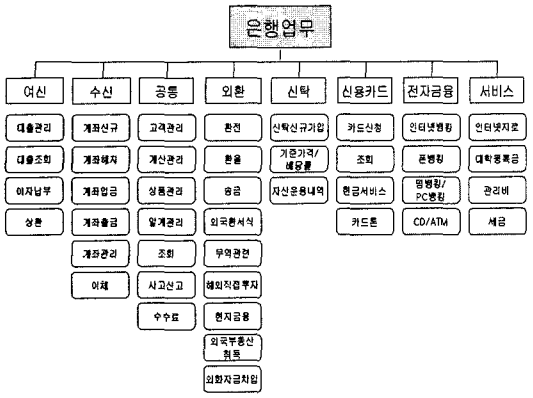

그림 7 은행 영역의 분류체계

행, 행정, 의료 3가지 영역 적용될 수 있는 컴포넌트가 될 수 있다. 따라서 본 논문에서는 이런 문제점을 해결하기 위해 범용성별 분류와 영역별 분류로 나누어서 분류체계를 정의한다. 영역에 따른 분류는 그림 6에 표현된 것처럼 범용성에 따른 분류 체계에서 다섯 번째 계층인 핵심 비즈니스 계층에 대해서 분류한다.

각각의 영역들에 대해서 다시 세부 분류 체계들을 정의하는데, 본 연구에서는 제시된 영역들 가운데 은행, 증권, 보험, 신용카드, 병원, 그리고 B2B 영역에 대해 소분류 체계들을 정의한다. 소분류 체계들 가운데 은행 영역의 소분류 체계 일부가 그림 7에서 표현되고 있다.

3.3 크기(Granularity)에 따른 분류

컴포넌트의 크기라 함은 어플리케이션 개발에 재사용되는 단위의 정도를 말한다. 현재 컴포넌트 크기에 대한 표준 규격이 마련되어 있지 않은 상태이다. 따라서 컴포넌트의 단위를 규정하는 시각들이 다양하다. 재사용되는 하나의 클래스를 컴포넌트라고 하거나, 여러 컴포넌트들로 구성된 프레임워크도 컴포넌트로 보거나, 또는 다른 컴포넌트를 포함하는 복합 컴포넌트를 컴포넌트로 보는

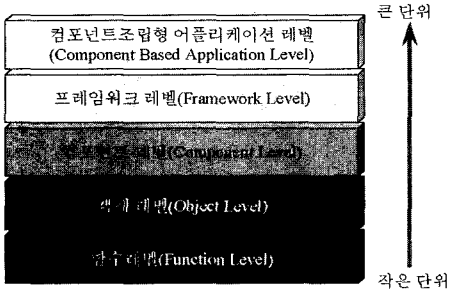

그림 8 크기에 따른 컴포넌트 분류 체계

경우가 있다[11,12,13]. 이처럼 컴포넌트의 크기들이 다를 수 있기 때문에 동일한 이름의 컴포넌트라고 하여도 크기가 다른 컴포넌트들이 된다. 이러한 문제점을 고려하여 컴포넌트를 크기에 따라 그림 8에 표현된 것처럼 분류한다.

함수 레벨의 컴포넌트는 독립적인 함수 단위로 구성된 컴포넌트를 의미한다. 이 레벨에 속하는 컴포넌트는 객체 단위가 아닌 기존의 절차적 언어로 구성된 재사용 가능한 함수로 개발된 컴포넌트를 의미한다. 객체레벨의 컴포넌트는 관련된 데이타와 함수를 갖는 독립적인 객체 단위로 개발된 단일 객체 혹은 복합 객체의 클래스이다. 예를 들어, 자바 언어로 개발된 클래스 수준이 객체 레벨에 속하는 컴포넌트가 된다. 컴포넌트 레벨은 기본적으로 컴포넌트 참조모델을 적용하여 개발된 컴포넌트들을 의미한다. 예를 들어, EJB 기반의 컴포넌트가 여기에 해당한다고 볼 수 있다. 따라서 이 레벨에 속하는 컴포넌트들은 표준 인터페이스를 반드시 제공해야 하며, 독립적으로 배포될 수 있는 단위이어야 한다. 프레임워크 레벨의 컴포넌트는 하나 이상의 컴포넌트들로 구성된 복합 컴포넌트에 해당하는 것으로서, 프레임워크 인터페이스와 핫 스팟(Hot Spot)을 제공해야 한다. 컴포넌트 조립형 어플리케이션 레벨은 재사용 가능한 컴포넌트들을 기반으로 해서 구축된 어플리케이션들을 의미한다. 따라서 이 레벨에 속하는 어플리케이션들은 일부 컴포넌트를 다른 컴포넌트로 대치가 가능하도록 독립적인 부품들로 구성되어야 한다.

3.4 추상화(Abstraction)에 따는 분류

추상화란 불필요한 것을 제거하고 사물의 본질을 분리해 내고 요약하여 핵심 정보만을 추출해 내는 것이다. 컴포넌트를 분류하는 데 있어서 이러한 추상화를 적용하는 이유는 컴포넌트에 대한 정의가 너무 다양하고 컴포넌트를 정의하는 수준이 넓은 의미를 가지고 정의하는 경우도 있고 좁은 의미를 가지고 정의하는 경우도 있다. 예를 들어 어떤 사람은 컴포넌트를 실제 시스템 구동시 배치되어 돌아가는 독립적인 모듈로 정의하는가 하면, 반면에 어떤 사람은 재사용할 수 있는 것은 모두 컴포넌트가 된다고 간주하는 경우도 있다. 후자의 경우를 보면 설계 패턴(Design Pattern)도 재사용이 되면 컴포넌트가 되고, 소스코드(Source Code)도 재사용이 되면 컴포넌트가 되는 것이다. 따라서 본 연구에서는 이러한 다양한 정의들을 반영하기 위해서 컴포넌트의 추상화 정도를 가지고 분류체계를 마련한 것이다. 추상화의 정도에 따른 분류 체계가 그림 9에 제시되어 있다.

패턴은 분석 패턴과 설계 패턴으로 구분되는데 분석

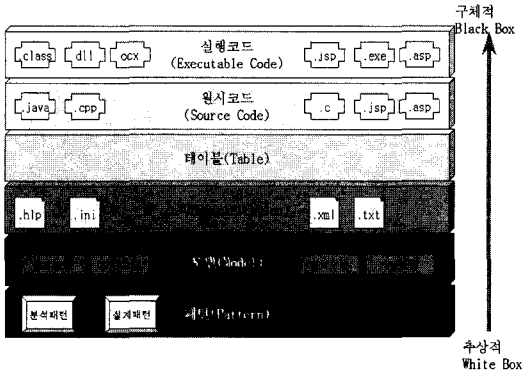

그림 9 추상화에 따른 컴포넌트 분류 체계

패턴은 개발 과정에서 분석 단계에 재사용될 수 있는 컴포넌트가 되고, 설계 패턴은 설계 단계에 적용될 수 있는 재사용 컴포넌트가 된다. 모델 컴포넌트 즉, 다이어그램들은 분석 혹은 설계 단계에서 재사용될 수 있는 컴포넌트가 된다. 스펙 컴포넌트들은 주로 설계 혹은 구현 단계에서 재사용될 수 있는 컴포넌트들이며, 테이블 컴포넌트들은 데이타베이스 설계 혹은 구현시에 재사용될 수 있는 단위의 컴포넌트이다. 원시코드 컴포넌트들은 기존에 주로 많이 사용했던 재사용 단위 컴포넌트들로서 소스 코드 단위 별로 재사용되는 컴포넌트들이다. 실행코드 컴포넌트들은 최근 컴포넌트 기술에서 초점을 두고 있는 단위의 컴포넌트들로서 구현이 다 되어있으면서 독립적으로 배치할 수 있는 수준의 컴포넌트들을 의미한다.

3.5 플랫폼(Platform)에 따른 분류

현재 컴포넌트 개발을 지원하는 컴포넌트 개발 플랫폼에는 썬사의 EJB와 JavaBean, OMG의 CORBA Component Model(CCM), 그리고 마이크로소프트사의 Component Object Model(COM)이 있는데 본 논문에서 플랫폼이라 함은 이러한 컴포넌트 기술 플랫폼외에 컴포넌트, 프레임워크, 또는 어플리케이션 개발에 적용되는 기술들을 총망라한다. 그 이유는 현재 운영되고 있는 컴포넌트 유통 사이트들을 보면 컴포넌트의 형태를 단지 배치(Deployment)할 수 있는 컴포넌트들 만을 취급하지 않고 자그마한 Graphic User Interface(GUI) Widget에서부터 어플리케이션에 이르기까지 다양한 레벨의 컴포넌트들을 유통하고 있다. 따라서 이러한 유형들을 모두 처리하기 위해서는 기술들을 모두 적용하여 분류를 해야 한다. 따라서 이러한 기술들에 대한 분류체계가 그림 10에 표현되어 있다.

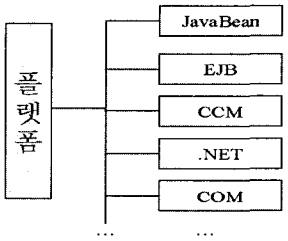

그림 10 플랫폼에 따른 컴포넌트 분류체계

3.6 언어(Language)에 따른 분류

언어에 따른 분류는 컴포넌트 개발시 사용된 언어를 따른 분류를 의미한다. 여기서 말하는 언어는 프로그래밍 언어만을 의미하는 것이 아니라 모델링 언어, 데이타베이스 질의어 등도 포함을 한다. 그 이유는 예를 들어 컴포넌트가 다이어그램이라고 하면 이 컴포넌트 개발시 사용된 언어는 프로그래밍 언어가 아닌 모델링 언어가 되기 때문이다.

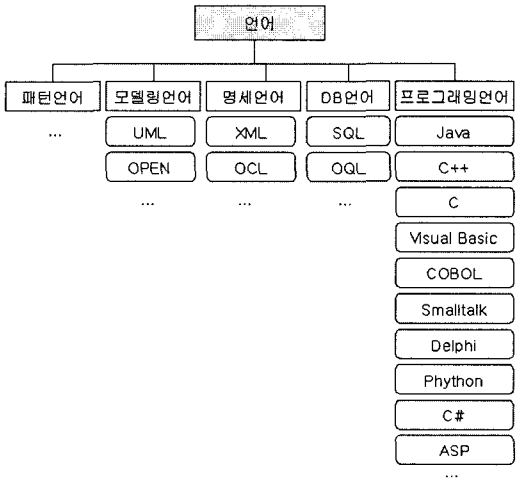

그림 11 언어에 따른 컴포넌트 분류체계

따라서 본 논문에서는 언어의 범위를 넓은 의미의 언어로 규정하고 있다. 이와 같이 언어에 따른 분류 체계가 그림 11에 제시되고 있다. UML[11]과 OPEN과 같은 것은 소프트웨어 분석 설계시에 적용되는 모델링 언어이고 XML이나 OCL과 같은 언어는 설계나 프로그램 또는 데이타에 대한 명세를 하는데 사용되는 언어이다.

4. BNF를 이용한 컴포넌트 분류코드 명세

이전 장에서 6가지 관점을 바탕으로 분류한 분류체계

들을 컴포넌트 유통시스템에서 효율적으로 관리하기 위해서는 분류코드가 필요하다. 특히 컴포넌트 분류 코드는 여러 컴포넌트 분류 체계들에서 해당 컴포넌트 분류 체계를 식별하기 위한 식별자 역할을 하기 때문이다. 본 논문에서는 이러한 분류 코드를 BNF 표기법을 이용하여 명세한다.

4.1 코드부여 원칙

컴포넌트 분류코드 부여 원칙은 다음과 같다.

첫째, 6가지 관점에 대한 분류 코드를 부여한다.

둘째, 각각의 관점별로 분류 코드를 부여한다. 여기서 관점에 따라 어떤 관점에서는 분류 단계가 대분류, 중분류, 소분류까지 나누어지는 경우가 있고, 어떤 관점에서는 분류 코드가 대분류나 중분류까지만 표현되는 경우가 존재한다.

셋째, 각 분류 체계마다 정의되지 않은 부분에 대해서는 'Ext'로 표현한다.

넷째, 각 관점과 관점별 분류 단계와의 구별은 ':'로 표현한다.

다섯째, 각 관점 별로 분류 단계는 대분류, 중분류, 소분류를 구별하기 위해 소수점을 찍어서 표현한다. 예를 들면, G:BS.FI.BK.XXX 형태로 표현한다.

4.2 분류코드 체계

먼저 컴포넌트는 6가지 관점에 대해 대분류 코드체계를 갖는다.

<Component>::=<G> <D> <S> <A> <P> <L>

여기서 G는 Generality의 시작문자를 의미하고, D는 Domain의 시작문자, S는 Size의 시작문자, A는 Abstraction의 시작문자, P는 Platform의 시작문자, L은 Language의 시작문자를 의미한다.

각각의 관점별로 분류한 코드 체계는 다음과 같다. 범용성 관점에서 분류한 코드 체계는 다음과 같이 구성된다.

<G>::=<UI>|<OP>|<SS>|<BG>|<BS>|<AP>

각각의 코드 명칭은 표 1에 표현된 것처럼 각각의 계층에 대한 영문자의 약어를 사용한다.

도메인 별로 분류한 코드 체계는 다음과 같이 구성된다.

표 1 계층별 분류 코드 의미

코드명	의미
UI	User Interface
OP	OS Platform
SS	System Service
BC	Business General
BS	Business Specific
AP	Application

<D>::=<FI> | <CO> | <HC> | <RE> | <GO> | <MA> | <EN> | <EC> | <TR> | <ED>

각각의 코드 명칭 부여 원칙은 도메인 명에 대해서 영문자의 약어를 이용한다. 그런데 도메인 명이 한 단어인 경우는 시작 문자 두문자를 이용해서 표현하고, 두 단어 이상인 경우는 단어의 시작문자를 조합해서 표현한다. 각각의 코드의 의미는 표 2에 표현되어 있다.

참고로 어떤 컴포넌트가 특정 도메인에 국한되지 않고 모든 도메인에 사용되는 경우는 'Ext'로 표현한다. 이렇게 분류된 각각의 도메인 분류 코드 별로 다시 서브 도메인 분류코드를 표현할 수 있다.

표 2 도메인별 코드 표기법

코드명	의미
FI	Finance
CO	Communication
HC	Health Care
RE	Real Estate
GO	Government
MA	Manufacture
EN	Entertainment
EC	Electronic Commerce
TR	Transportation
ED	Education

예를 들어 FI 도메인에 대한 중분류 코드 체계는 다음과 같이 구성된다.

<FI>::=<BK> | <AR> | <SC> | <CC>

여기서 BK는 Banking, AR은 Assurance, SC는 Security, CC는 Credit Card를 의미한다.

크기별로 분류한 코드체계는 다음과 같다.

<S>::=<FN> | <OJ> | <CP> | <FW> | <CA>

여기서, FN은 Function, OJ는 Object, CP는 Component, FW는 Framework, CA는 Component-based Application을 의미한다.

추상화 수준 별로 분류한 코드체계는 다음과 같다.

<A>::=<PT>|<MD>|<SP>|<TB>|<SC>|<EC>

여기서 PT는 Pattern, MD는 Model, SP는 Specification, TB는 Table, SC는 Source Code, EC는 Executable Code를 의미한다.

플랫폼 별로 분류한 코드 체계는 다음과 같다.

<P>::=<JAB>|<EJB>|<NET>|<COM>|<CCM>

여기서 JAB는 JavaBean, EJB는 Enterprise Java Bean, NET은 .Net, COM은 MS의 COM, CCM은 CORBA Component Model을 의미한다.

언어별로 분류한 코드 체계는 다음과 같다.

<L>::=<PTL> "," {<PTL>} <MOL> "," {<MOL>} <SPL> "," {<SPL>} "," <DBL> "," {<DBL>} "," <PGL> "," {<PGL>}

여기서 PTL은 Pattern Language, MOL은 Modeling Language, SPL은 Specification Language, DBL은 Database Language, PGL은 Programming Language를 의미한다.

언어별로 분류한 각각의 분류별로 다시 소분류를 나누면 다음과 같다.

<MOL>::=<UML>|<OPEN>
<SPL>::=<XML>|<OCL>
<DBL>::=<SQL>|<OQL>
<PGL>::=<Java>|<C++>|<C>|<C#>|<VB>|<ASP>|<Delphi>|<Phython>|<Smalltalk>|<COBOL>

5. 실험 및 평가

본 장에서는 본 논문에서 제안한 분류 체계와 분류 코드 표기법을 적용한 실무 사례들을 제시하고 기존의 분류체계들과 적중율과 정확도를 가지고 평가하고자 한다. 본 연구에서는 컴포넌트들은 UI 컴포넌트에서부터 비롯하여 비즈니스 컴포넌트에 이르기까지 다양한 컴포넌트들을 본 컴포넌트 분류체계에 적용하여 실험했다. 여러 다양한 컴포넌트 유형들 가운데 일부 몇 개에 대해 예를 들어 적용 사례를 제시하고자 한다.

5.1 실험 1

사용자 인터페이스 컴포넌트의 일례로서 'Toggle Button'이란 컴포넌트를 가지고 본 논문에서 제시한 분류 체계에 적용하면 다음과 같다.

- 범용성 관점에서는 'User Interface 계층(UI)의 Button 계층'에 속한다.
- 도메인 관점에서는 모든 도메인에 적용되기 때문에 'Ext'에 해당한다.
- 크기 관점에서는 이 컴포넌트는 단일 클래스로 만들어졌으나 컴포넌트 표준 모델인 JavaBean에 따라서 만들었기 때문에 '컴포넌트수준(CP)'에 해당한다.
- 추상화 관점에서는 이 컴포넌트는 배포시 클래스 파일 형태로 배포하기 때문에 '실행코드(EC) 수준에서 클래스 파일'에 해당한다.
- 플랫폼 관점에서는 이 컴포넌트는 JavaBean을 이용해서 만들어졌기 때문에 '자바빈(JAB)'에 해당한다.
- 언어 관점에서는 이 컴포넌트는 Java 언어를 이용해서 만들어졌기 때문에 '프로그래밍 언어/Java'에 해당한다.

이러한 분류 체계를 제시한 분류코드를 적용하여 표현하면 다음과 같이 표현된다.

'ToggleButton'=<G:UI.Button><D:Ext><S:CP><A:EC.Class><P:JAB><L:PGL.Java>로 표현된다.

이를 도식적으로 표현하면 그림 12와 같다.

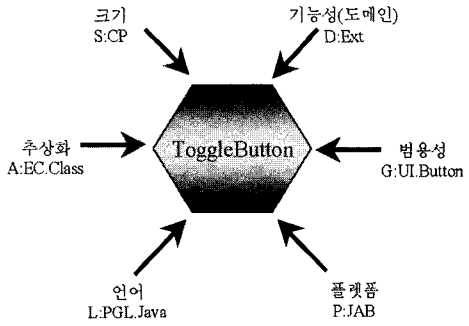

그림 12 'ToggleButton'에 대한 분류 코드 표현

5.2 실험 2

파일 Upload하는 기능을 가진 'FileManager'라는 컴포넌트를 적용하면 다음과 같다.

- 범용성 관점에서는 'System Service 계층(UI)의 Utility 계층의 File Service 계층'에 속한다.
- 도메인 관점에서는 모든 도메인에 적용될 수 있기 때문에 'Ext'에 해당한다.
- 크기 관점에서는 이 컴포넌트는 복합 클래스로 만들어졌기 때문에 '객체수준(OJ)'에 해당한다.
- 추상화 관점에서는 이 컴포넌트는 배포시 실행코드 형태로 배포하기 때문에 '실행코드(EC) 수준에서 exe 파일'에 해당한다.
- 플랫폼 관점에서는 이 컴포넌트는 특정 컴포넌트 표준에 따라서 개발되지 않았기 때문에 'Ext'에 해당한다.
- 언어 관점에서는 이 컴포넌트는 Java 언어를 이용해서 만들어졌기 때문에 '프로그래밍 언어/Java'에 해당한다.

이러한 분류 체계를 제시한 분류코드를 적용하여 표현하면 다음과 같이 표현된다.

'FileManager'=<G:SS.UT.FP><D:Ext><S:OJ><A:EC.exe><P:Ext><L:PGL.Java>로 표현된다.

이를 도식적으로 표현하면 그림 13과 같다.

5.3 실험 3

전자 상거래의 '주문관리'를 담당하는 기능을 가진 'Order Management'라는 컴포넌트를 적용하면 다음과 같다.

- 범용성 관점에서는 '핵심 비즈니스 계층(BS)'에 속한다.

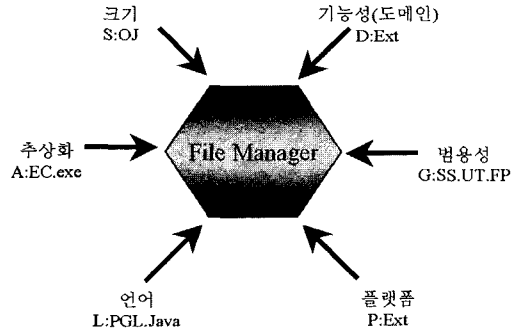

그림 13 'FileManager' 컴포넌트에 대한 분류코드 표현

- 도메인 관점에서는 '전자상거래(EC)'에 속한다.
- 크기 관점에서는 이 컴포넌트는 복합 클래스이면서 EJB 컴포넌트 표준에 의거하여 만들어졌기 때문에 '컴포넌트 수준(CP)'에 해당한다.
- 추상화 관점에서는 이 컴포넌트는 배포시 JAR 코드 형태로 배포하기 때문에 '실행코드(EC) 수준에서 JAR 파일'에 해당한다.
- 플랫폼 관점에서는 이 컴포넌트는 EJB 표준에 따라서 개발되었기 때문에 'EJB'에 해당한다.
- 언어 관점에서는 이 컴포넌트는 Java 언어를 이용해서 만들어졌기 때문에 '스펙언어(SPL)/XML'+'프로그래밍 언어(PGL)/Java'에 해당한다.

이러한 분류 체계를 제시한 분류코드를 적용하여 표현하면 다음과 같이 표현된다.

'OrderManagement'=<G:BS><D:EC><S:CP><A:EC.jar><P:EJB><L:SPL.XML+PGL.Java>로 표현된다.

이를 도식적으로 표현하면 그림 14와 같다.

5.4 평가

이 절에서는 본 논문에서 제시한 분류체계와 기존의 분

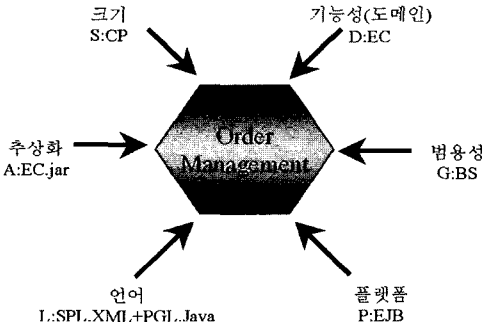

그림 14 'OrderManagement' 컴포넌트에 대한 분류코드 표현

류체계를 비교 평가하기 위해서 적중율(Hit Ratio), 정확도(Correctness), 복잡도(Complexity), 유연성(Flexibility) 등과 같은 척도들을 가지고 비교하고자 하다.

적중율이라 함은 등록하고자 하는 전체 컴포넌트들 가운데서 분류체계에 얼마나 많이 적중되는지를 백분율로 측정하는 기준이다. 그 공식은 다음과 같다. 적중율이 높을수록 분류체계가 잘 정의된 것이다.

- 적중율 = 적중되는 컴포넌트 개수 / 등록하고자 하는 컴포넌트 개수 * 100%

정확도라 함은 고객이 찾고자 하는 컴포넌트에 대한 요구사항 개수들 가운데서 이 요구사항과 일치하여 검색된 컴포넌트의 개수를 백분율로 측정하는 기준이다. 그 공식은 다음과 같다. 정확도가 클수록 분류체계가 잘 정의된 것이다.

- 정확도 = 일치하여 검색된 컴포넌트의 개수/검색하고자 하는 컴포넌트에 대한 요구사항 개수 * 100%

복잡도라 함은 고객이 특정 컴포넌트를 검색하거나 등록하는데 명세해야 할 항목의 수를 측정하는 기준이다. 항목의 수가 많을수록 분류체계의 복잡도가 크다.

유연성이라 함은 고객이 특정 컴포넌트를 검색하는데 필요한 옵션의 수를 측정하는 기준이다. 즉, 검색하는데 한가지 경로로만 제공한다면 불필요한 컴포넌트들이 검색되어질 수 있을 뿐만 아니라 고객의 만족도를 떨어뜨릴 수 있다. 그러나 유연성이 크다는 것은 그만큼 검색 옵션의 수가 많다는 것이므로 여러가지 경로를 제공함으로써 고객이 원하는 컴포넌트가 보다 정확하게 검색될 수 있다. 그러나 이런 경우 복잡도는 높아질 수도 있다.

이러한 측정 도구들을 가지고 기존의 분류체계와 본 연구에서 제시한 분류체계를 비교한 도표가 그림 15와 그림16에 표현되어 있다. 이외에도 복잡도와 유연성 등 기타 척도들을 바탕으로 여러 분류체계들과 비교한 결과가 표 3에 제시되어 있다.

표 3 기존 분류체계와 6View 기반 분류체계와의 비교표

분류체계(시스템) 비교항목	Component Source	Flashline	ImagiCom	ETRI	6View 기반 분류체계
복잡도	낮다	낮다	낮다	낮다	높다
검색의 유연성	낮다	낮다	낮다	높다	높다
분류체계 기준	기능중심	기능중심	기능중심	도메인 중심	6가지 관점
검색의 편리성	높다	높다	높다	낮다	낮다

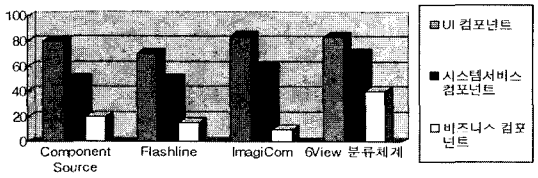

그림 15 적중율을 통한 비교 도표

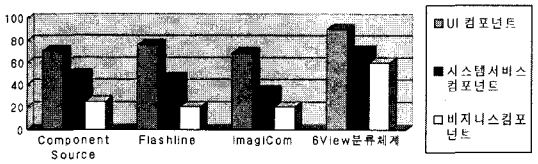

그림 16 정확도를 통한 비교 도표

6. 결 론

본 논문에서는 효율적인 컴포넌트 유통시스템을 구축하는 데 필요한 컴포넌트 분류체계를 6가지 관점에서 정의하여 제시하였다. 특히 본 논문에서 제시하고 있는 6가지 관점은 컴포넌트의 효율적인 등록, 검색, 관리를 위해 고안된 관점이다. 기존의 단일 관점 중심에서 분류한 분류체계에 비해 본 논문에서 제시한 분류체계는 여러가지 각도를 제시한 분류체계를 정의함으로써 컴포넌트들을 효율적으로 등록할 수 있을 뿐만 아니라 고객이 원하는 정확한 컴포넌트가 검색될 수 있도록 지원한다. 또한 본 논문에서 제시하는 분류체계는 객체지향 개념을 적용한 계층적 분류체계이기 때문에 새로운 유형의 분류 항목이 생기더라도 추가나 확장이 용이하게 이루어질 수 있다. 향후 연구과제로는 본 연구에서 제시한 여러 관점별 분류체계들을 추가 및 확장하는 것이다.

참 고 문 헌

[1] ComponentSource(httP://www.componentsource.com)
[2] Flashline (http://www.flashline.com)
[3] D. Chappell, "On COM ActiveX vs. Java Beans," Object Magazine, January 1998.
[4] Sun Microsystems, *Enterprise JavaBeans Specification*, at URL: http://www.javasoft.com, 1999.
[5] Microsoft Crop., *The Component Object Model Specification*, Microsoft Press, 1995.
[6] ImagiCom (http://www.imagicom.com)
[7] IBM, "Architecture of the San Francisco Frameworks," (http://www.research.ibm.com/journal/sj/ 372/bohrer.html), Dec., 1997.
[8] Booch G., Rumbaugh J., and Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
[9] 한국전자통신연구원, "영역별 컴포넌트 분류방법에 관한 연구", 1999.
[10] Object Management Group, *CORBA Components*, December 1998.
[11] Desmond F. D'souza and Alan C. Wills, *Objects, Components, and Frameworks with UML*, p91-234, Addison-Wesley, 1998.
[12] Szyperski C., *Component Software: Beyound Object-Oriented Programming*, Addison Wesley Longman, Reading, Mass., 1998.
[13] Butler Group, "Component-Based Development: Application Delivery and Integration Using Componentised Software," September, 1998.

조 은 숙
1993년 동의대학교 전산통계학과 이학사. 1996년 숭실대학교 컴퓨터학과 공학석사. 2000년 숭실대학교 컴퓨터학과 공학박사. 1999년 ~ 2000년 ㈜객체정보기술 과장. 2000년 9월 ~ 현재 동덕여대 정보학부 데이타정보학과 강의전임교수. 2002년 1월 ~ 현재 한국전자통신 연구원 초빙연구원. 관심분야는 컴포넌트기반 소프트웨어 개발 방법론, 소프트웨어 아키텍처, 컴포넌트 정형명세, 컴포넌트 메트릭

이 종 국
1991년 고려대 물리학과 이학사. 1993년 고려대 물리학과 석사. 1995년 ~ 1999년 10월 대우정보 시스템 연구원. 2000년 ~ 현재 숭실대학교 컴퓨터학과 박사과정 재학중. 관심분야는 컴포넌트 개발 방법론, 컴포넌트 정형명세, 소프트웨어 아키텍처, Web Service

김 수 동
1984년 Northeast Missouri State University 전산학 학사. 1988년 The University of Iowa 전산학 석사. 1991년 The University of Iowa 전산학 박사. 1991년 ~ 1993년 한국통신 연구개발단 선임연구원. 1993년 ~ 1994년 The University of Iowa 교환교수. 1994년 ~ 1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월 ~ 현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 컴포넌트 개발 방법론, 객체지향 개발 방법론, 분산 시스템, 컴포넌트 정형명세, 소프트웨어 시험