룰 기반 분석패턴을 사용한 비즈니스 컴포넌트 방법

(Business Component Method using a Rule-Based Analysis Pattern)

이 용 환 † 민 덕 기 ††
(Yong Hwan Lee) (Duck Ki Min)

요 약 기존 소프트웨어 개발 프로세스는 분석 단계보다는 설계나 구현 단계만을 강조하고 있으며 서브시스템이 많은 복잡한 비즈니스 애플리케이션 분석 시 분석가의 경험이나 업무 지식의 차이에 따라 스타일과 추상화 레벨이 다른 분석 산출물을 작성하게 된다. 이러한 분석 산출물은 산출물 일관성이나 가독성에 영향을 미친다. 본 논문에서는 많은 서브시스템들에 대한 분석을 서로 다른 분석가에 의해서 수행해야 하고 외부 이벤트에 대해 트랜잭션 처리를 룰 기반으로 처리해야 하는 도메인 상에서 객체 기반의 중요 개념을 룰 기반으로 추출하고 그들 간의 상호작용 모델링을 효과적으로 할 수 있는 룰 기반 분석 패턴을 제시한다. 또한 제시한 분석 패턴이 가지는 3개의 핵심 개념들을 기반으로 UML Components 개발 프로세스 상에서 비즈니스 컴포넌트를 개발하는 방법을 은행 수신업무 적용 사례를 통해 제시한다.

키워드 : 분석패턴, 객체지향 모델링, 개념적 모델링, 컴포넌트 기반 개발 프로세스, 룰 기반

Abstract The existing CBD development methods deal with the analysis phase in a superficial manner. Applying such a superficial analysis to business applications with a number of subsystems makes analysis models be inconsistent with levels and styles, only depending on experiences of the analysts. This inconsistent analysis might cause more serious problems during the subsequent development phases, resulting in the failure of the projects. In this paper, we propose a rule-based analysis pattern that provides an analysis template for business applications. This pattern analyzes the concepts of business applications by using external events and internal rules that process the events. Employing this pattern, a huge business application can be developed by a couple of co-analysts who work together in a consistent and systematic manner. This paper also describes an efficient way to develop business components with the suggested analysis pattern using banking deposit case study through UML Components development process.

Key words : Analysis Pattern, Object-Oriented Modeling, Conceptual Modeling, CBD Development Process, Rule-Based, UML Components

1. 서 론

기존 소프트웨어 개발 프로세스[1,2]는 주로 분석 단계(Analysis Phases)를 피상적으로 다루고 있으며 설계나 구현 단계를 강조하고 있어 분석 단계에서 요구 사항이 충분히 반영되어 있는지 검토하지 않고 설계나 구현으로 넘어간다[3]. 많은 업무 분석가들이 서로 다른 서브시스템들에 대해 분석 모델을 수행하는 경우 스타일과 추상화 레벨이 다른 분석 산출물(예: 클래스 다이어그램 형태의 비즈니스 객체 모델)을 작성하게 된다. 시스템 분석을 위한 틀이 없이 분석, 설계 그리고 구현을 수행 했을 때 각각 다른 레벨의 산출물을 작성하게 되며 이는 산출물 가독성을 저하시키고 개발 생산성을 떨어뜨린다. 특히 분석 단계에서의 서로 다른 개념들은 프로젝트 위험(Risk)을 증가시킬 수 있다.

최근 소프트웨어 엔지니어 분야 연구자들은 특정 도메인 별 특성들에 대한 요구사항이나 개념들을 식별하기 위한 분석 패턴들을 연구하고 있다[4-6]. 하지만 외부 이벤트에 대해 내부 프로세스를 룰 기반으로 처리하

· 본 논문은 2006년도 건국대학교 학술활동지원금에 의한 논문임
† 정 회 원 : 동덕여자대학교 컴퓨터학과 교수
yhlee@konkuk.ac.kr
†† 정 회 원 : 건국대학교 정보통신공 교수
dkmin@konkuk.ac.kr
논문접수 : 2005년 8월 19일
심사완료 : 2005년 12월 21일

는 비즈니스 어플리케이션을 위한 분석 패턴은 존재하지 않는다. 또한 기존의 분석 패턴들에 대한 연구들은 분석 단계에 대해서만 강조를 하고 있지 분석 패턴상의 주요 개념들이 컴포넌트 개발 프로세스 상의 분석, 설계 그리고 구현까지 어떻게 체계적으로 연결되어 적용되는지에 대한 연구는 없다.

본 논문에서는 시스템 외부 이벤트에 대해 비즈니스 어플리케이션 트랜잭션을 룰 기반으로 처리해야 하는 도메인에서 시스템 내부의 비즈니스 개념들을 룰(Rule) 기반으로 추출하고 그들 간의 상호작용 관계를 효과적으로 모델링 할 수 있는 분석 패턴을 제시한다. 또한 제시한 분석 패턴이 가지는 몇 개의 핵심 개념이 UML Components개발 프로세스 상의 분석, 설계 그리고 구현 단계까지 어떻게 연결되어 컴포넌트를 개발하는지에 대해 은행 수신업무 예제를 통해 제시한다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 분석 패턴에 대한 관련 연구를 제시하고 3절에서는 본 논문에서 제시한 룰 기반 분석 패턴에 대해서 기술한다. 다음으로 4절에서는 룰 기반 분석 패턴을 UML Components 개발 방법론상에 어떻게 적용되는지를 은행 수신업무 예제를 통해 기술하고 5절에서는 은행 CBD 프로젝트에 적용한 효과를 기술하고 마지막으로 6절에서 결론 및 향후 과제를 기술한다.

2. 관련연구

2.1 배경 및 관련 분석패턴

분석단계는 문제영역을 이해하는 것이 초점이기 때문에 문제영역을 잘 반영한 개념모델을 작성해야 한다. 이러한 개념모델에는 단지 요구사항 나열만이 아니라 요구사항 달성을 위한 내부 원리까지 포함한다. 모델에 있어 옳고 그름은 없으며 모델 평가는 얼마나 더 유용하느냐에 의해서 결정된다. 분석 패턴은 실제 특정 도메인에 유용하고 다른 곳에서도 유용할 것으로 생각되는 아이디어로서 개념모델 작성 시 문제 영역 이해, 특정 도메인 상의 문제 해결을 위한 내부 원리 이해와 같은 목적을 위해 분석틀을 제시한다.

소프트웨어 개발 프로세스 초기 단계에서 패턴을 사용하고자 하는 많은 다른 접근 방법이 존재한다[7]. 예를 들면, 최근에 Fowler[8]는 회계, 무역, 조직 구성에 대한 추상화 같은 비즈니스 프로세스의 개념적인 모델링을 표현하기 위해 사용할 수 있는 몇 가지 패턴을 확인했다. 또한 각 도메인 별 특징을 반영하기 위한 패턴[9,10]들을 연구하고 있는데 예를 들면 임베디드 시스템(Embedded System) 도메인 상의 어떤 특징을 반영하기 위한 분석 패턴인 객체 분석 패턴[11]도 존재한다.

Gross와 Yu[12]는 비기능적 요구사항과 설계패턴과의 관계에 대해 연구했고 Robertson[13]은 요구사항 패턴을 식별, 정의 그리고 접근하기 위해 이벤트/유스케이스 모델링 사용에 대해 제안했다. 또한 Sutchliffe[14]는 서로 다른 어플리케이션들에 대한 일반적인 요구사항을 식별하기 위해 유스케이스 시나리오를 어떻게 조사해야 하는지에 대해서 기술했다.

Fernandez와 Yuan[6]은 시멘틱 분석 패턴을 제시했다. 시멘틱 분석 패턴은 몇 개의 유스케이스 혹은 소수의 요구사항 집합들을 가지고 있으며 상태 다이어그램과 시퀀스 다이어그램 관점에서 제안된 솔루션의 동적인 행위를 묘사하고 있다. 마지막으로 van Lamsweerde는 KAOS[15] 방식을 개발했다. 이 방식은 메타모델과 이를 통해 목적 기반의 초기 요구사항 획득을 어떻게 하는지에 대해서 기술했다. 이 밖에도 소프트웨어 아키텍처 패턴[16], 데이타베이스 접근 패턴[17], 무정지형 통신 시스템 패턴[18], 웹서비스에서 비 동기 호출을 위한 패턴[19,20]등이 제안되었다.

기존에 제시했던 분석 패턴과 비교해 본 논문에서 제시한 분석패턴은 먼저 비즈니스 컴포넌트를 개발할 때 액터로부터의 모든 외부 이벤트에 대해 룰기반 형태의 비즈니스 프로세스 처리를 위한 주요 개념과 그들 간의 관계를 효과적으로 표현할 수 있다는 것이다. 두 번째로 기존 분석 패턴이 개발 프로세스 상의 분석 단계에 중점을 두고 있는 반면에 본 논문에서 제시한 분석 패턴은 패턴에서 제시한 몇 개의 핵심 개념을 사용해 UML Components 상에서 어떻게 효과적인 컴포넌트 모델링을 수행하는지를 제시한다는 것이다.

2.2 비즈니스 컴포넌트 재사용 기법

컴포넌트 기반 소프트웨어 재사용 기법은 수정, 조립 그리고 개조로 구분된다[21]. 컴포넌트 수정은 컴포넌트의 기본 특성중의 하나인 프라퍼티(Property)에 의해 이루어지는 컴포넌트의 변경과정을 의미한다. 프라퍼티란 컴포넌트의 범위와 행위를 결정짓는 특성으로 컴포넌트를 재 사용하여 어플리케이션을 설계할 때 그 값을 변경할 수 있다. 컴포넌트 수정을 통한 재사용은 컴포넌트 내부 구현에 대한 이해를 요구하기 때문에 재사용에 어려움이 많다. 컴포넌트 조립은 컴포넌트를 재사용하기 위한 가장 자연스러운 과정으로 컴포넌트들끼리 합성(Composition)하는 경우와 다른 어플리케이션들과 통합(Integration) 하는것을 의미한다.

컴포넌트 개조는 컴포넌트가 어떤 이유로 미리 정의된 인터페이스나 행위를 변경해야 할 경우를 의미한다. 간단하게는 인터페이스 이름 변경이나 파라미터의 형식이 변경되는 것을 들 수 있으며 복잡하게는 새로운 인터페이스를 추가하는 경우도 있다. 이러한 컴포넌트 개조를 위한 방법으로는 Adapter기법, Wrapper기법[22],

Superimposition기법[23], Binary Adaptation기법[24] 등이 있다. 이러한 개조 기법은 컴포넌트 독립성을 이용하는 것으로 변경 영향 범위를 변경 원인이 되는 컴포넌트에게만 국한 시키는 방식이다. 하지만 컴포넌트 독립성만으로 소프트웨어 확장성 및 유연성을 보장하기에 미비하다는 제한점을 가지고 있다.

비즈니스 요구사항은 시스템 개발중에도 개발 완료 후에도 지속적으로 변할 수 있는 부분이다. 만일 가변적인 부분이 비즈니스 컴포넌트안에 포함된다면 변경이 발생할 때 마다 코드를 수정해야 한다. 따라서 가변적인 부분과 공통되는 부분을 분리하여 컴포넌트를 개발하고 가변적인 부분을 별도로 재정의 할 수 있도록 하는 것이 컴포넌트 재사용성을 강화할 수 있다.

룰은 일반적으로 하나 또는 그 이상의 비즈니스 프로세스의 행위를 통제하는 규칙으로서 비즈니스 프로세스적인 룰과 비즈니스 도메인 룰로 구성된다. 비즈니스 도메인 룰은 대상이 가지고 있는 가변적 특성과 특성을 해석하는 가변적 방법을 정의한다. 예를 들면 고객의 나이를 구하는 도메인 서비스는 문제의 특성에 따라 주민번호에 의한 법적 나이 일수도 있고 은행에서 사용하는 나이일 수도 있다. 비즈니스 프로세스 룰은 하나의 업무를 처리하는데 필요한 작업종류, 순서, 처리 조건을 정의하는 룰을 말한다.

기존에 제시했던 분석 패턴과 다르게 본 연구에서 제시한 룰 기반 분석 패턴을 사용해 컴포넌트를 개발할 때는 컴포넌트의 재사용성이나 적응성을 높일 수 있는 장점을 가지고 있다. 이유는 제시한 룰 기반 분석 패턴을 통해 식별된 별도의 룰 컴포넌트가 비즈니스 도메인 룰과 비즈니스 프로세스 룰을 처리할 수 있기 때문이다.

3. 룰 기반 분석 패턴

3.1 개요(Synopsis)

제시한 룰 기반 분석 패턴의 용도는 먼저 많은 서브시스템들을 가진 복잡한 시스템을 서로 다른 분석가에 의해 분석 된 후 비즈니스 컴포넌트를 개발할 경우 산출물 일관성을 유지하고 시스템의 가독성을 좋게 함으로서 개발 생산성이나 유지보수를 향상시키기 위해서 사용한다. 두 번째는 외부 액터에 의해 발생된 모든 이벤트에 대해 비즈니스 어플리케이션 트랜잭션을 룰 기반으로 처리할 경우 시스템 내부의 비즈니스 개념들을 룰 기반으로 추출하고 그들 간의 상호작용 관계를 효과적으로 모델링 할 때 사용한다. 세 번째로 제시한 룰 기반 분석패턴이 가지는 핵심 개념들을 사용해 효과적으로 비즈니스 컴포넌트를 개발할 수 있으며 식별된 컴포넌트중에 룰 컴포넌트를 사용해서 비즈니스 도메인 룰과 비즈니스 프로세스 룰을 처리함으로서 컴포넌트의 재사용성이나 적응성을 높일 수 있다.

3.2 배경(Context)

제시한 룰 기반 분석패턴이 해결하고자 하는 첫 번째 문제는 서로 다른 업무 분석가들이 각 서브시스템을 분석한 후에 이를 기반으로 컴포넌트를 개발할 경우 각 업무 분석가의 경력이나 배경 지식에 따라 각 서브시스템 분석 내용이나 세부 레벨이 다른 산출물을 작성하게 된다. 내용이나 레벨이 다른 분석 산출물을 기반으로 컴포넌트를 개발할 경우 컴포넌트 개발이나 유지보수 측면에서 많은 비효율성이 발생하며 이는 프로젝트 관리에 많은 위험을 초래한다.

두 번째 문제는 컴포넌트의 재사용성과 적응성이다. 기존 컴포넌트 재사용에 대한 연구는 환경과 요구사항 변경에 대한 파급효과를 최소화하기 위한 컴포넌트 독립성에 대한 것이다. 하지만 컴포넌트 독립성 만으로는 만족할 만한 컴포넌트의 변경 용이성이나 재사용성을 달성할 수 없으며 룰 컴포넌트와 같은 별도의 컴포넌트를 통해서 비즈니스 도메인 룰과 비즈니스 프로세스 룰을 처리해야 한다. 제시한 룰 기반 분석 패턴은 가변적인 것들을 룰 컴포넌트에서 관리한다는 측면에서 이러한 문제점에 대한 해결책이 될 수 있다.

3.3 고려사항들(Forces)

논문에서 제시한 분석 패턴이 제기한 문제에 대한 솔루션을 내기 위해서는 몇 가지 제약사항이 존재한다. 먼저 제시한 분석패턴이 적용될 수 있는 도메인은 은행업무와 같이 모든 외부 이벤트에 대해 룰 기반으로 거래를 처리해야 하는 경우이다. 두 번째로 분석 패턴상의 Target은 적용 영역에 따라 한 개 이상의 Target들이 존재할 수 있으며 각 Target별 액션을 위한 룰 들이 존재한다. 세 번째로 Target에 대한 상태 명세는 액션의 대상이 되는 Target이 생명주기를 가지며 생명주기 동안 각 상태를 분별할 수 있을 때 가능하다. 네 번째로 제시한 룰 기반 분석 패턴을 사용해 컴포넌트를 개발할 때 컴포넌트 재사용성을 강화하기 위해서는 비즈니스 프로세스적인 룰이나 비즈니스 도메인 룰에 대한 처리는 룰 컴포넌트 내부에 포함시켜야 한다. 이는 환경, 요구사항 그리고 기능 변경 발생 시 컴포넌트 재사용성과 적응성을 향상시키기 위함이다.

3.4 해결책(Solution)

그림 1은 본 논문에서 제시한 분석 패턴에 대한 아키텍처 템플릿이다. 제시한 분석 패턴이 가지는 주요 개념들에 대한 설명을 위해 은행 고객이 ATM을 사용해 돈을 예치(Deposit)한다고 가정하자. 고객이 예치 서비스 요청을 보내면 ATM은 어떤 타입의 요청이 발생했는지를 식별하고 이 요청 이벤트 타입에 설정된 룰들을 찾아서 해당 조건을 검사한 후에 룰에 따라 고객의 계좌

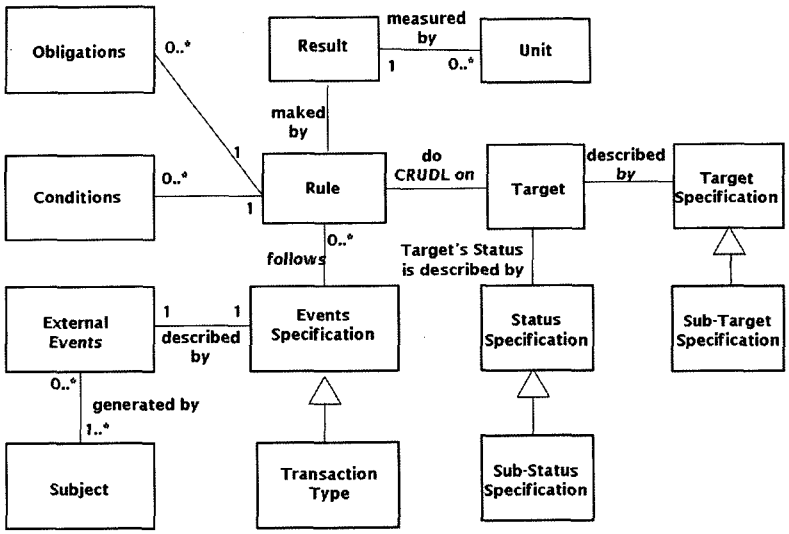

그림 1 룰 기반 분석 패턴 아키텍처 템플릿

를 대상으로 액션을 수행한다. 액션 수행 후의 결과 값은 룰에 따라 고객 계좌에 다시 재 반영된다. 다음은 각 개념들이 가지고 있는 주요 책임에 대한 설명이다.

3.4.1 Subject

그림 1에서 주체(Subject)는 서비스 사용을 위해 외부 이벤트를 발생시키는 액터를 말한다. 주체가 하는 책임은 시스템에 거래요청을 함으로서 이벤트를 발생시킨 후에 처리결과를 받는 것이다. ATM 예제에서 은행 고객이 바로 주체가 된다.

3.4.2 External Events, Events Specification

외부이벤트들은 시스템 외부 액터가 시스템을 사용함으로서 발생하는 모든 이벤트를 말한다. ATM예제에서는 은행고객이 요청하는 입금, 출금, 계좌이체와 같은 거래들이 외부 이벤트가 된다. 이벤트 명세는 이벤트에 대한 메타 개념(Meta Concept)으로서 이벤트에 대한 표준 명세를 담고 있다. 이벤트 명세에는 발생 가능한 이벤트 타입뿐만 아니라 각 이벤트 타입 별로 따라야 할 룰을 가지고 있다.

3.4.3 Rule, Conditions, Obligations

외부 액터에 의해 발생한 모든 이벤트는 비즈니스 룰에 적합한 지 검사된 후에 처리된다. 비즈니스 룰은 이벤트 표준 명세상의 이벤트 타입(Event Type)과 관련해서 정의되며 룰에 대한 표현은 조건(Condition), 의무(Obligation)사항 그리고 액션(Action)의 형태로 기술된다. 의무사항은 룰 컨디션들이 만족되고 액션 수행 전에 해야 할 행위들로서 주로 액션수행과 관련된 룰 관련 정보들을 얻어오는 것이다.

3.4.4 Result, Unit

결과 개념은 프로세스 관점에서 어떤 행위를 의미하며 정보 관점에서는 액션 수행 후의 상태 결과를 나타낸다. 액션 수행 여부와 방법은 비즈니스 룰에 의해서 결정되며 액션 수행후의 상태 결과에는 소스에 해당하는 이벤트 타입 정보를 담고 있다. ATM예제에서 결과는 이자율, 세금 그리고 적수 계산과 같은 것들이다. 액션 수행 후 결과 값은 룰을 통해 원래의 대상 자원에 다시 반영한다. 단위(Unit) 개념은 Folwer의 Quantity Pattern[8]을 룰 기반 분석 패턴에 적용한 것으로 대상 자원이나 결과를 양적으로 측정하기 위한 기본 단위를 나타낸다.

3.4.5 Target, Target Specification, Sub-Target Specification

외부 이벤트에 대해 룰의 컨디션이 만족되면 룰은 어떤 행위를 하게 된다. 이러한 행위는 대상이 존재하는데 Target은 행위의 근거가 되는 대상을 의미한다. Target은 프로세스 적인 것 보다는 정보를 표현하는 엔티티이며 CRUD(Create, Read, Update, Delete)를 수행한다. 대상자원 명세는 대상자원이 가질 수 있는 다양한 타입들을 표현하며 그것들의 표준 명세를 제공한다. 명세에서 정의한 타입에 따라 다양한 Sub-Target Specification이 존재한다. 앞의 ATM 예제를 들어 설명하면 대상 자원인 계좌는 대상자원 명세에서 명시한 타입에 따라 다양한 계좌 상품(예: 요구 불 계좌, 정기예금 계좌, 당좌계좌)이 가능하다.

3.4.6 Status Specification, Sub-Status Specification

대상자원 상태에 대한 명세는 Target이 일정한 생명주기를 가지면 생명주기상의 특정 상태 구분이 가능한

경우에 적용된다. Status Specification은 Target의 상태에 대한 명세이며 Sub-Status Specification은 Status Specification에서 명시한 다양한 종류의 서브상태를 나타낸다. ATM예제에서 정상계좌, 비정상 계좌가 여기에 해당되면 비정상 계좌는 해지, 사고 그리고 세금면제 계좌와 같은 것들이 존재한다.

3.4.7 룰기반 이벤트 처리과정

그림 2는 룰 기반 분석 패턴을 사용해 비즈니스 프로세스를 처리하는 단계이다.

먼저 주체는 이벤트 명세에 명시된 외부 이벤트를 발생한다. 룰은 자신이 관리하는 가변적인 정보를 사용해 룰에 명시된 조건을 검사한다. 명시된 조건이 만족되면 룰은 액션을 수행하기 전에 수행해야 할 의무사항이 있는지 검사한다. 만일 의무사항 내용이 액션수행에 필요한 Target관련 정보를 가져오는 경우에 룰은 Target에 대해 CRUD를 요청한다. 룰은 의무사항 수행 후에 룰에 명시된 액션이 존재한 경우 Result에 액션을 위임한다. 룰은 액션 수행 후 Result로부터 반환된 값을 받아 조건을 검사한 후에 원래의 대상 자원에 다시 반영하다. 룰은 Result와 Target 흐름을 통제하는 조정자 역할을 수행한다.

3.5 결과(Consequence)

본 논문에서 제시한 룰 기반 분석 패턴은 다음과 같은 장점들을 제시한다. 첫 번째로 다른 분석 패턴과 같이 비즈니스 프로세스 분석을 위한 개념적 틀을 제시하지만 본 논문에서는 액터로부터 발생하는 외부 이벤트에 대해 룰 기반 프로세스 분석을 위한 틀을 제시한다. 둘째로 명세(Specification)에 해당하는 지식 개념(Knowledge Concept)들과 운영 레벨에 해당하는 실제 개념(Real Concept)을 분리함으로써 각 도메인 별 혹은 환경 별로 실제 개념에 해당하는 추가적인 이벤트나 대상 자원의 타입들을 쉽게 추가할 수 있다. 세 번째로 대상 자원이나 결과 개념을 양적으로 측정하기 위해 필요한 단위(Unit) 개념을 추가함으로써 대상 자원을 다양한 단위로 측정할 수 있다. 네 번째로 대상 자원이 일정 생명주기를 가지고 각 생명주기 상태가 구분(은행 계좌 예: 정상 계좌 혹은 비정상 계좌)이 가능하고 미리 정의되어 있는 경우에는 대상자원의 상태를 상속을 이용해 서브 타입 형태로 상태의 개념을 확장해서 표현할 수 있다. 다섯째 룰 기반 분석 패턴은 비즈니스 어플리케이션 프로세스를 룰, 결과, 대상의 3가지 주요 개념 관계로 표현하기 때문에 개발자들이 쉽게 실제 모듈이나 컴포넌트가 가지는 작업(Task)의 복잡성을 단순화 시킬 수 있으며 개발 프로세스상의 주요 단계들을 효과적으로 수행할 수 있다. 특히 여러 서브시스템이 존재하는 상황에서 모델러들이 같은 관점에서 일관되고 효율적으로 컴포넌트 모델링을 수행 할 수 있다. 마지막으로 제시한 룰 기반 분석 패턴을 사용해 컴포넌트 개발 시 프로세스나 정보상의 가변적인 부분들을 룰 컴포넌트를 통해 관리함으로써 컴포넌트 재사용성이나 적응성을 향상 시킬 수 있다.

4. 은행 수신업무를 통한 UML Components 개발 프로세스 적용 예제

4절에서는 3절에서 제시한 룰 기반 분석 패턴이 가지는 3가지 개념(Rule, Target, Result)을 사용해 UML Components 개발 프로세스[25]상에서 비즈니스 컴포넌트를 개발하는 방법을 은행 수신업무 예제를 통해 제시

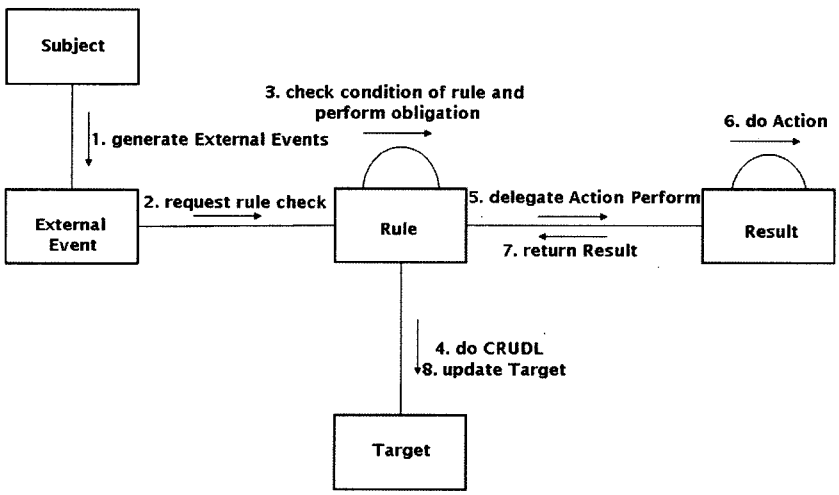

그림 2 룰 기반 분석 패턴의 이벤트 처리 과정

한다. UML Components 개발 프로세스는 크게 요구사항 정의, 컴포넌트 명세, 컴포넌트 공급, 컴포넌트 조립, 테스트 그리고 배포의 과정으로 이루어져 있다. 컴포넌트 명세 단계는 컴포넌트 식별을 위한 분석 단계이다. 본 논문에서는 UML Components 개발 프로세스 상에서 유스케이스 모델, 비즈니스 객체 모델, 인터페이스와 컴포넌트 식별 그리고 컴포넌트 아키텍처 다이어그램과 같은 CBD 주요 산출물에 대해서만 기술한다.

4.1 은행 수신업무

은행 수신 업무(Deposit)는 모든 종류의 예금계좌 관리와 그에 따른 모든 거래를 처리하는 개인 고객을 상대로 한 은행 소매업의 한 부분으로서 타 업무(예: 고객관리, 회계처리) 부분과 많은 인터페이스를 갖고 있다. 그림 3은 비즈니스 프로세스와 비즈니스 도메인 차원에서 수신 업무를 분류한 것이다.

은행 수신 업무 프로세스는 크게 계좌 관리, 계좌처리, 통장관리 업무로 구분된다. 계좌관리는 계좌 생성, 수정, 해지 그리고 이자 관리로 구성되며 계좌처리는 입금, 출금, 조회 그리고 이체로 구성된다. 통장관리는 통장 발행과 통장 출력으로 구성된다

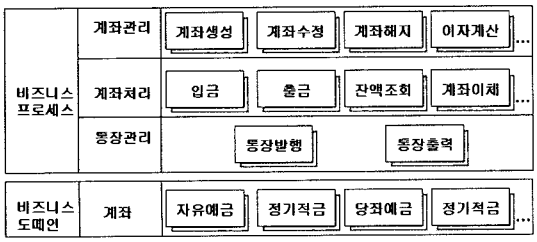

그림 3 은행 수신서브시스템

비즈니스 도메인 차원에서 수신 업무와 관련된 예금은 크게 분류해 보면 유동성 예금과 고정성 예금으로 분리된다. 유동성 예금에는 입출금이 자유롭지만 이자율이 낮은 보통 예금, 이자는 없고 주로 수표발행과 관련해 기업들이 사용하는 당좌예금, 보통 예금보다 이율은 조금 높고 주로 개인이 사용하는 계좌로서 저축예금 등이 있으며 고정식 예금에는 일정기간을 정해놓고 가입하며 이자가 가장 많은 정기예금, 일정 기간마다 부금을 납입하고 만기에 한번에 받는 정기적금, 정기예금이긴 하지만 통장을 발행하지 않고 증서를 발행하여 그 증서를 양도나 매매 할 수 있고 만기에 그 증서를 가진 사람이 은행에서 돈을 받을 수 있는 양도성 예금, 만기가 없는 정기예금인데 대신 해약하기 몇 일 전에 은행에 통보해야만 지급받을 수 있는 통지 예금 등이 존재한다. 이러한 계좌의 개설은 예금 종류별로 이루어지며 상품 종류별로 존재하는 룰에 따라 이자율, 세율, 이전계좌 사용여부, 복수계좌 사용 여부가 결정된다.

4.2 유스케이스와 비즈니스 객체 모델

UML Components 분석 단계 주요 산출물은 유스케이스 모델과 비즈니스 객체 모델이다. 룰 기반 분석패턴에서 제시한 주체, 외부 이벤트 그리고 이벤트 명세 개념들은 각각 액터, 유스케이스, 유스케이스 시나리오를 추출하기 위해 사용한다. 그리고 3절에서 설명한 룰 기반의 분석 패턴은 클래스 다이아 그램 형태의 비즈니스 객체 모델을 작성하기 위한 분석틀을 제공한다.

그림 4는 본 논문에서 제시한 분석 패턴을 사용해 수신 업무 분석가가 작성한 비즈니스 개념 모델이다. 그림

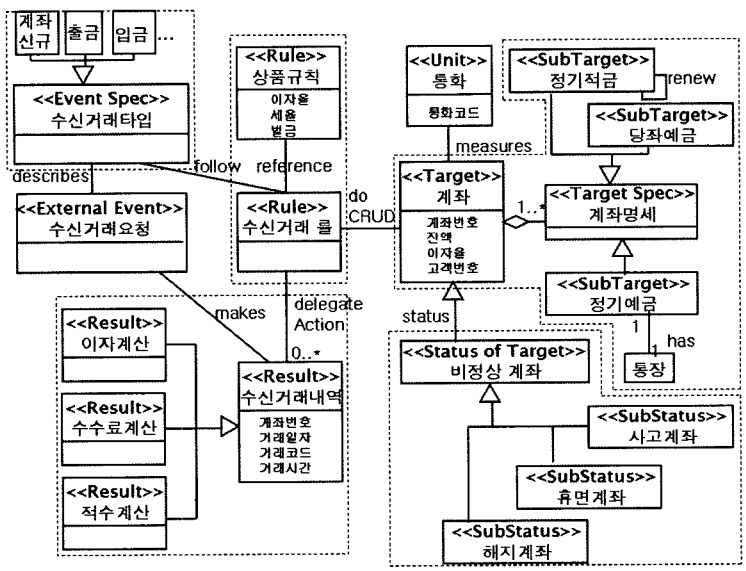

그림 4 수신 업무 비즈니스 개념 모델

4와 같이 업무 분석가는 분석패턴이 제시한 업무 내용 범위와 추상화 레벨에 따라 업무 분석을 수행하게 된다. 따라서 많은 서브시스템을 각각 다른 분석가에 의해서 분석을 수행하더라고 추상화 레벨이 일치된 분석 산출물을 작성할 수 있다.

그림 5는 본 논문에서 제시한 분석패턴상의 개념과 수신 업무 분석가가 분석한 실제 인스턴스와의 매핑관계를 제시하고 있다.

수신 업무에서 룰 개념은 계좌 상품별로, 대상자원 개념에 대한 인스턴스는 계좌로, 서브 대상 자원 개념은 정기적금이나 당좌예금 등과 같은 계좌상품 형태로, 서브 상태 개념은 수신업무에서는 비정상적인 계좌인 해지계좌나 사고계좌 형태로 나타난다.

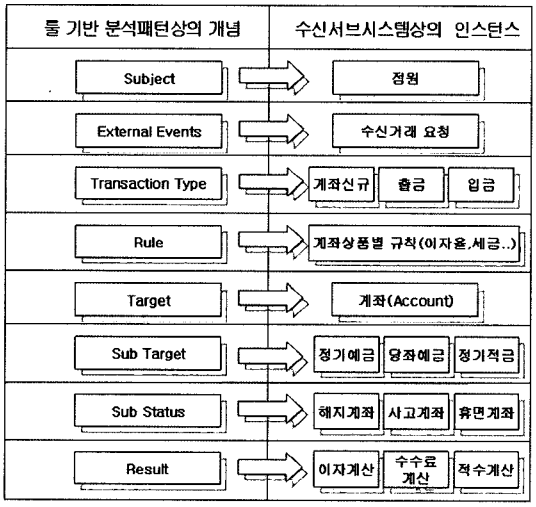

그림 5 패턴상의 개념과 수신업무 인스턴스와의 매핑

4.3 인터페이스와 컴포넌트 식별

UML Components는 인터페이스를 시스템 인터페이스와 비즈니스 인터페이스로 구분한다. 시스템 인터페이스는 사용자와 시스템 사이의 상호작용을 위한 유스케이스 단계(Step) 로직을 제공하고 있으며 유스케이스에서 사용자와 시스템 사이의 상호작용 시나리오로부터 도출되며 주요 역할은 외부 이벤트를 처음으로 받아들이는 파사드(Facade)와 다른 서브 시스템 내부에 존재하는 시스템 인터페이스와 상호작용을 조정한다.

비즈니스 인터페이스는 비즈니스 어플리케이션의 핵심 로직들을 구현한다. 비즈니스 인터페이스 식별을 위해 먼저 분석 단계에서 식별된 비즈니스 객체모델을 기반으로 프로세스 처리와 관련해 다른 객체와의 의존성이 없이 독자적으로 존재할 수 있는 객체를 핵심 타입(Core Type)[12]으로 식별한다. 그리고 각 핵심 타입별로 한 개의 비즈니스 인터페이스를 할당한다.

본 논문에서 제시한 룰 기반 분석 패턴을 사용해 비즈니스 개념모델을 작성한 후 이를 기반으로 핵심 타입을 식별할 경우 핵심 타입은 Rule, Target 그리고 Result이다. 나머지 개념들은 핵심 타입을 위한 종속 개념들이다. 따라서 세 개의 핵심 타입에 대해 각각 Rule, Target 그리고 Result 인터페이스와 같은 세 개의 비즈니스 인터페이스가 식별된다.

그림 6은 수신 서브 시스템에서 비즈니스 인터페이스를 도출하기 위한 핵심 타입과 비즈니스 인터페이스 식별을 나타내고 있다. Rule, Target 그리고 Result와 같은 3개의 핵심 타입에 각각 IDepRuleMgr, IDepAccountMgr, IDepTrsResultMgr과 같은 3개의 인터페이스가 식별된다. 그리고 수신 서브 시스템상의 모든 유스케이스에 대해 한 개의 시스템 인터페이스 IDEPSIMgr를 식별한다. 점선 형태의 블록은 이 인터페이스가 관리해야 할 정보의 범위를 나타내는 정보모델을 나타낸다.

인터페이스 식별 후 인터페이스와 컴포넌트간 매핑을 사용해 초기 컴포넌트 타입을 식별한다. 그림 7은 인터페이스와 컴포넌트간 네 가지 매핑을 나타내고 있다. 첫 번째 매핑은 컴포넌트 별로 하나의 인터페이스를 두는 방식이다. 두 번째는 하나의 컴포넌트 안에 여러 개의 인터페이스가 존재하는 경우이다. 세 번째는 여러 개의 컴포넌트가 한 개의 인터페이스를 공유하는 방식으로 인터페이스 구현은 각각 다르다. 네 번째는 외부에서는 한 개의 컴포넌트가 한 개의 인터페이스를 가지고 있는 첫 번째와 유사하지만 내부에서만 사용할 수 있는 M개의 컴포넌트가 존재하는 형태이다.

초기에는 그림 7의 1:1 기본 매핑을 사용해 식별된 각 인터페이스 별로 하나의 컴포넌트 타입을 식별한다. 즉, 시스템 인터페이스에 시스템 컴포넌트를 할당하고 비즈니스 인터페이스인 룰, 대상자원 그리고 결과 인터페이스에 각각 룰 컴포넌트, 대상자원 컴포넌트, 결과 컴포넌트 타입을 할당한다.

기본적으로 한 개의 인터페이스에 한 개의 컴포넌트를 할당하지만 인터페이스나 컴포넌트가 가지는 규모(예: 단위컴포넌트, 컴포넌트 연합), 특성(예: 특정 도메인 전용, 조직간 공통) 그리고 구현(예: 비즈니스 엔티티, 비즈니스 프로세스, 비즈니스 유틸리티 컴포넌트)에 따라 여러 개의 인터페이스나 컴포넌트로 세분화되며 따라서 인터페이스와 컴포넌트 간 매핑은 그림 7에서 제시한 다양한 매핑을 통해 세부적인 컴포넌트를 식별할 수 있다.

그림 8은 룰 기반 분석 패턴이 가지는 주요개념(Rule, Target, Result)을 사용해 인터페이스와 컴포넌트를 식별할 때 인터페이스와 컴포넌트간 매핑 관계를 제시한

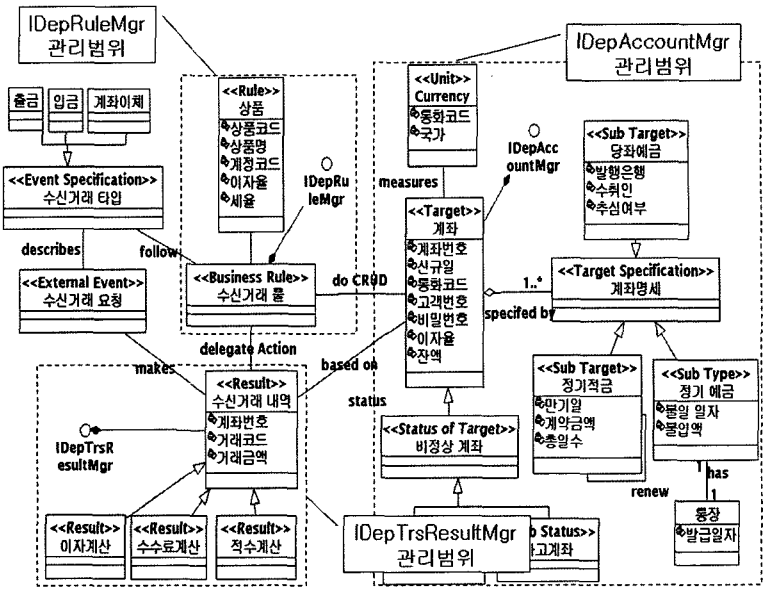

그림 6 수신 서브시스템의 핵심 타입과 비즈니스 인터페이스 식별

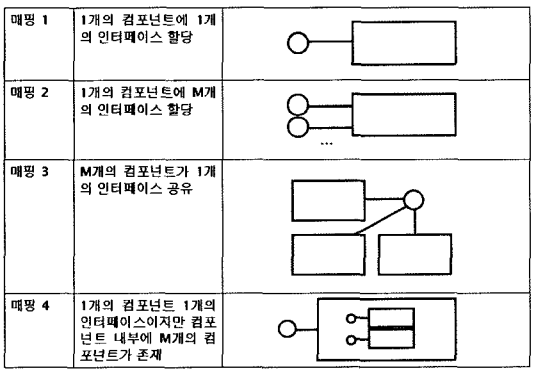

그림 7 인터페이스와 컴포넌트간의 매핑 종류

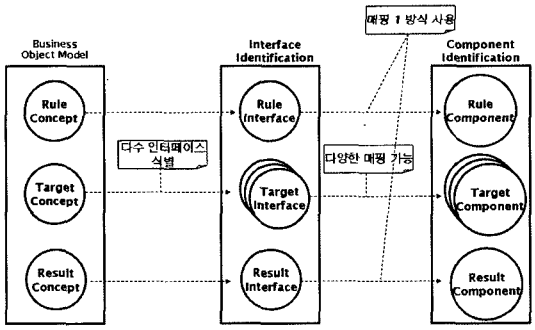

그림 8 룰 기반 분석 패턴 사용 시 인터페이스와 컴포넌트간의 매핑

다. 룰과 결과는 그림 7의 매핑 1 방식을 사용한다. 하지만 대상자원 개념은 대상자원 명세나 혹은 상태 명세에 따라 다양한 대상 자원 타입으로 확장될 수 있다. 따라서 여러 개의 비즈니스 인터페이스나 컴포넌트가 가능하며 둘 사이에 다양한 매핑 관계가 존재할 수 있다.

각 서브 시스템 별로 식별된 컴포넌트 타입들은 한 개의 시스템 인터페이스를 가진 시스템 컴포넌트와 룰 기반 분석 패턴이 가지는 세 개의 주요 개념을 통해 식별된 비즈니스 컴포넌트들이 존재한다. 첫 번째 비즈니스 컴포넌트는 룰 개념에서 파생된 한 개의 룰 비즈니스 컴포넌트이고 두 번째는 대상 자원 개념에서 파생된 M개의 대상자원 비즈니스 컴포넌트가 존재하며 마지막으로 결과 개념에서 파생된 한 개의 결과 비즈니스 컴포넌트가 존재한다.

그림 9는 수신 서브시스템에 대해 초기 1: 1매핑을 수행하고 대상자원 인터페이스를 세분화 한 후에 그림 7에서 제시한 4가지 매핑을 적용한 예제이다. 초기 식별된 인터페이스를 기반으로 1:1 기본 매핑을 수행한다. 즉, DepositManager시스템 컴포넌트에 IDEPSIMgr 시스템 인터페이스를 할당했으며 비즈니스 인터페이스는 각각 DepositRule에 IDEPRuleMgr를, DepositAccount Target에 IDEPAccountMgr를, DepositTransaction-Result에 IDEPTrsResultMgr를 할당했다.

기본 매핑이 완료된 후에 정제된 매핑을 수행하는데 시스템 컴포넌트, 룰과 결과 비즈니스 컴포넌트는 기본 매핑을 그대로 유지한다. 하지만 대상자원 비즈니스 컴포넌트의 경우에는 대상자원 명세에 따라 여러 종류의 서브 대상자원이 가능하기 때문에 여러 개의 인터페이스와 컴포넌트가 가능하며 다양한 매핑 관계가 가능하다.

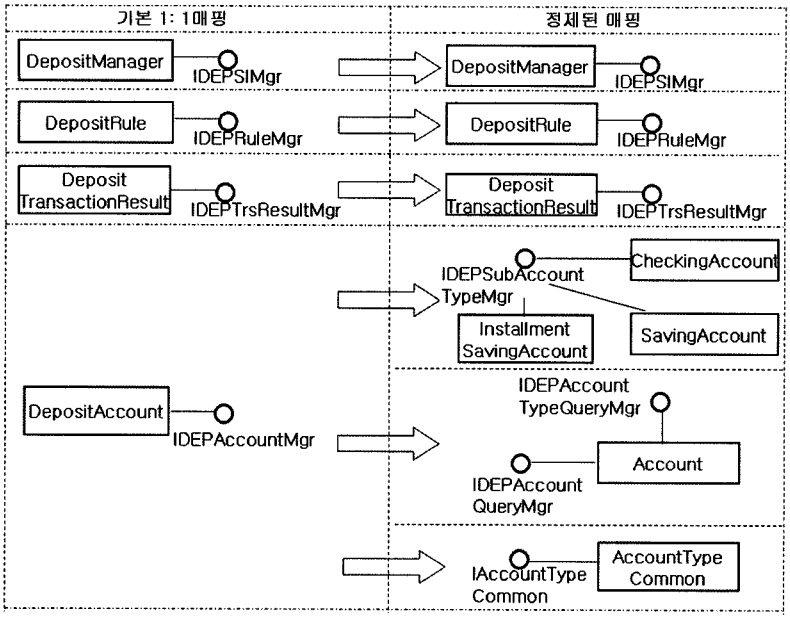

그림 9 수신서브 시스템 인터페이스와 컴포넌트간의 매핑

그림 9의 정제된 매핑을 보면 DepositAccount 대상자원 컴포넌트는 CheckingAccount, SavingAccount, InstallmentSavingAccount 같은 계좌 상품별 컴포넌트들로 세분화 되었으며 이들은 모두 하나의 IDEPAccountMgr 비즈니스 인터페이스를 공유한다. 이는 여러 개의 컴포넌트가 한 개의 인터페이스를 공유하는 1: M 매핑을 보여주는 것으로서 각 계좌 상품들이 공통으로 제공해야 하는 인터페이스이지만 인터페이스를 구현할 때는 각 계좌 상품별로 다르게 구현되는 방식이다.

DepositAccount 컴포넌트는 계좌 자체에 대한 서비스를 제공하는 Account비즈니스 컴포넌트로 세분화 되었다. Account컴포넌트는 계좌 자체에 대한 서비스를 제공하는 인터페이스와 계좌 상품별 컴포넌트에서 필요로 하는 서비스를 제공하는 인터페이스로 세분화 되며 각각 IDEPAccountQueryMgr, IDEPSubAccountTypeMgr로 명명된다. 이 매핑은 한 개의 컴포넌트가 여러 개의 인터페이스를 갖는 M:1 매핑 관계를 보여주고 있다.

초기 DepositAccount 대상자원 컴포넌트는 또한 모든 계좌 상품들의 공통로직을 제공하는 AccountTypeCommon로 세분화 되었으며 IAccountTypeCommon인터페이스를 가진다. 이 방식은 그림 7의 4번째 매핑방식을 보여주는 것으로 각 계좌 상품별 컴포넌트에서만 사용되는 컴포넌트로 외부에서는 각 계좌 상품별 컴포넌트가 1개의 인터페이스를 갖고 있는 것처럼 보이지만 실제로 내부에서는 M개의 컴포넌트가 존재하는 형태이다.

4.5 컴포넌트 아키텍처 다이어그램

그림 10은 룰 기반 분석 패턴을 통해 식별된 컴포넌트 타입들을 기반으로 작성된 컴포넌트 아키텍처 다이아 그램이다. 룰, 대상자원 그리고 결과 비즈니스 컴포넌트는 본 논문에서 제시한 룰 기반 분석 패턴을 통해 식별된 비즈니스 컴포넌트 타입 들이다.

룰 컴포넌트가 대상자원 컴포넌트와 결과 컴포넌트를 조정하는 컨트롤로 역할을 수행하며 대상자원과 결과 컴포넌트 타입이 모델 역할을 수행한다. 파사드 컴포넌트 타입은 룰, 대상자원 그리고 결과 컴포넌트들을 외부 액터로부터 숨김으로써 재사용성을 강화하기 위해 만든

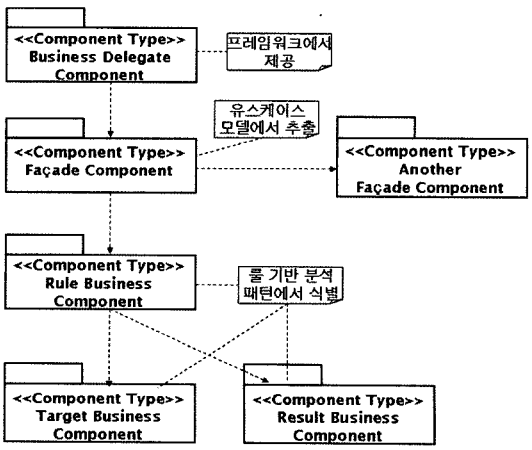

그림 10 컴포넌트 아키텍처 다이어그램

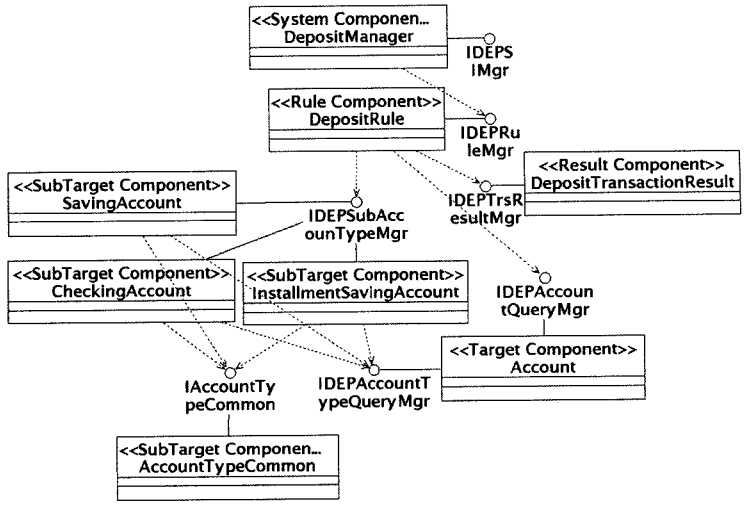

그림 11 수신 서브시스템 컴포넌트 아키텍처 다이어그램 예제

컴포넌트 타입이다. 그리고 하나의 거래를 처리하기 위해 다른 서브 시스템상의 비즈니스 서비스가 필요할 경우 그 서브시스템 내부의 파사드 컴포넌트를 호출한다.

Business Delegate 컴포넌트 타입은 전체 시스템에 한 개만 존재하는 컴포넌트 타입으로 분석이나 설계 단계에서 만들어지는 것이 아니라 프레임워크 구축 단계에서 생성된다. 이 컴포넌트 타입은 모든 서비스 요청이 처음으로 들어가는 유일한 진입 점으로서 로그 처리, 예외처리 그리고 날짜 계산과 같은 유스케이스들이 공통으로 처리해야 하는 작업을 수행한다. 그리고 비즈니스 계층에 전달된 요청을 처리하기 위해 파사드 컴포넌트를 찾아 해당 메소드를 호출한다.

그림 11은 그림 10에서 제시한 컴포넌트 아키텍처 다이어그램을 기반으로 작성한 수신 서브시스템의 컴포넌트 아키텍처 다이어그램 예제이다.

수신 서브시스템상에서 제공하는 서비스를 사용하는 모든 클라이언트는 시스템 컴포넌트인 DepositManager가 가지고 있는 인터페이스인 IDEPSIMgr를 호출함으로써 서비스를 제공받는다. DepositRule 비즈니스 룰 컴포넌트는 대상자원 컴포넌트와 결과 컴포넌트를 조정한다. 따라서 대상자원 컴포넌트와 결과 컴포넌트간에는 직접적인 의존 관계가 존재하지 않으며 또한 비즈니스 프로세스나 도메인에 관련된 룰들은 룰 컴포넌트에 포함시킴으로써 컴포넌트 재 사용성이나 변경 용이성 향상을 기할 수 있다. 각 계좌 상품별 컴포넌트들은 한 개의 비즈니스 인터페이스를 공유하고 있으며 계좌 상품들의 공통로직을 구현한 AccountTypeCommon 컴포넌트는 계좌 상품 별 컴포넌트에서만 사용 가능하며 또한 계좌 자체에 대한 서비스를 제공하는 Account컴포넌트는 2개의 인터페이스를 가지고 있는데 IDEPAccount-QueryMgr는 룰 컴포넌트에서 사용되며 IDEPAccount-TypeQueryMgr는 계좌상품 컴포넌트 들에서만 사용된다.

Rule, Target 그리고 Result컴포넌트 타입들은 다시 크게 비즈니스 프로세스적인 것과 비즈니스 도메인적으로 구분이 된다. 비즈니스 프로세스 컴포넌트 타입은 단위 도메인 별 비즈니스 서비스 업무 흐름과 처리 조건을 정의하여 비즈니스 트랜잭션을 처리하는 컴포넌트이다. 비즈니스 도메인 컴포넌트는 비즈니스 도메인 별 단위 서비스를 제공한다. 구현단계에서 식별된 컴포넌트를 EJB로 구현할 때 먼저 시스템 컴포넌트인Business Facade 타입의 컴포넌트는 무 상태 세션 빈 타입으로 구현한다. 세 개의 비즈니스 컴포넌트 각각은 EJB로 구현할 때 비즈니스 프로세스는 무 상태 세션 빈으로 구현하며 비즈니스 도메인은 Entity Bean을 사용해서 구현된다. 즉, 룰 컴포넌트, 대상자원 컴포넌트, 결과 컴포넌트는 EJB의 무 상태 세션 빈 형태로 구현된다. 컴포넌트 재사용성이나 적응성을 향상시키기 위해서는 룰 컴포넌트를 룰 엔진 형태로 구현함으로써 가변적인 룰들을 룰 컴포넌트 내부가 아닌 외부에서 관리할 수도 있다.

5. 은행 CBD 프로젝트에 적용한 효과

한국의 C 은행 국외점포 업무를 컴포넌트 형태로 개발하는 실제 CBD프로젝트에 본 논문에서 제시한 룰 기반 분석 패턴을 적용했다. C은행의 컴포넌트 개발 프로젝트는 전체 서브시스템의 개수가 56개이며 분석 설계를 위해 20명의 모델러들이 각각 3개의 서브 시스템을 분석 및 설계 했다.

20명의 모델러들은 제시한 룰 기반 분석 패턴이 가지는 몇 개의 개념들을 기반으로 자신이 맡은 업무를 분석했기 때문에 상당히 직관적으로 분석 산출물을 작성할 수 있으며 설계 동안에도 분석패턴에서 제시한 핵심 개념들이 인터페이스나 컴포넌트에도 그대로 적용되었기 때문에 컴포넌트 설계를 쉽게 할 수 있을 뿐만 아니라 관련된 사람들과 대화가 용이했다. 최종적으로 모델러들은 자신이 맡은 서브 시스템 별로 기본적으로 비즈니스 파사드 컴포넌트, 룰 컴포넌트, 대상자원 컴포넌트, 결과 컴포넌트를 도출했으며 대상 자원 컴포넌트는 대상 자원 명세와 대상자원 상태 명세에 따라 여러 개의 서브 컴포넌트들이 식별되었다. 이렇게 해서 전체 56개 서브시스템에 대해 300개 정도의 컴포넌트를 룰 기반 분석 패턴을 적용해 식별해서 구현했다.

설계 이후 구현은 EJB로 구현하였으며 모델러들의 기술 성숙도에 문제가 있어 모델에 참여하지 않은 별도의 EJB코더(Coder)들이 구현했다. 컴포넌트 모델링에 참여하지 않았던 EJB 코더(Coder)들도 본 논문에서 제시한 룰 기반 분석 패턴상의 개념을 통해 직관적으로 시스템을 이해해서 구현에 참여할 수 있었다.

6. 결론 및 향후 과제

본 논문에서는 룰을 기반으로 비즈니스 내부 개념들과 이들 사이의 관계를 추출하기 위한 개념적인 분석틀을 제시했으며 또한 제시한 분석 패턴이 가지는 주요 개념들을 기반으로 UML Components 개발 프로세스상에 적용하는 방법을 은행 수신업무 예제를 통해 기술했다.

제시한 분석 패턴의 장점은 먼저 하나의 시스템 안에 여러 개의 서브시스템을 가지는 복잡한 비즈니스 어플리케이션 상황에서 각각의 분석가들이 룰 기반 분석 패턴을 통해 표준화되고 일관된 비즈니스 개념들을 식별할 수 있다. 두 번째로 분석패턴이 제시한 주요 개념들을 사용해 UML Components와 같은 CBD개발 프로세스상에서 효과적으로 비즈니스 컴포넌트를 개발 할 수 있으며 산출물의 가독성과 일관성을 좋게 한다. 세 번째로 비즈니스 프로세스적인 룰이나 비즈니스 도메인 룰과 같은 가변적인 부분들을 별도의 룰 컴포넌트에서 관리하도록 함으로서 컴포넌트 재사용성이나 적응성을 높일 수 있다.

향후 연구로는 본 논문에서 제시한 룰 기반 분석 패턴을 사용해 식별된 룰 컴포넌트를 룰 엔진 기반으로 구현 함으로서 가변적인 비즈니스 컴포넌트 개발을 가능하게 해 보고 또한 다양한 응용 영역에 적용해 보아야 할 것이다.

참 고 문 헌

[1] D'souza D.F. and Wills A.C., Objects, Components, and Components with UML, Addison-Wisely, 1998.

[2] F.P. Brooks. The Mythical Man-Month: Essays on software engineering, Anniversary Edition. Addison-Wesely, 1999.

[3] C.D. Sigwart, G.L.V. Meer, and J.C. Hansen, Software Engineering: A project oriented Approach. J. Leisy Jr., Franlin, Beedle, and Assoc., 1990.

[4] L.A. Campbell, "Enabling Integrative Analyses and Refinement of Object-Oriented Models with Special Emphasis on High-Assurance Embedded Systems," PhD thesis, Michigan State Univ., East Lansing, 2004.

[5] S. Konard, B.H.C. Cheng, L.A. Campbell, and R. Wassermann, "Using Security Patterns to Model and Analyze Security Requirements," Proc. Requirements for High Assurance Systems Workshop (RHAS '3), Sept. 2002.

[6] E.B. Fernandez and X. Yuan, "Semantic Analysis Patterns," Proc. 19th Int'l Conf. Conceptual Modeling(ER 2000), pp.183-195, 2000.

[7] A. Geyer-Schulz and M Hashler, "Software Engineering with Analysis Patterns," 2001,http://wwwai.wu-wien.ac.at/~hahsler/research.

[8] M. Fowler, Analysis Patterns: Reusable Object Models. Addison-Wesley, 1997.

[9] E. Gamma, R. Helm, R Johnson, J. Vlissides," Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[10] H.Gomaa, Designing Concurrent, Distributed, and Real-Time Application with UML, Addison-Wesley, 2000.

[11] S.Konrad, Betty H.C. Cheng, Laura A, Campbell, "Object Analysis Patterns for Embedded Systems," IEEE Transaction on software engineering, Vol.30, No.12, December 2004.

[12] D.Gross and E.S.K. Yu, "From Non-Functional Requirements to Design through Patterns," Requirements Eng., Vol.6, No.1, pp.18-36, 2001.

[13] S. Robertson, "Requirements Patterns via Events/Use Cases," 1996, http://www.systemsguild.com/GuildSite/SQR/Requirements_Patterns.html.

[14] A.G. Sutcliffe, N.A. Maiden, S. Minocha,and D. Manuel, "Supporting Scenario-Based Requirements Engineering," Software Eng., Vol. 24, No. 12, pp. 1072-1088, Dec. 1998.

[15] A.Dardenne, A.van Lamsweerde, and S. Fickas, "Goal-Directed Requirements Acquisition," Selected Papers Sixth Int'l Workshop Software Specification and Design, pp.3-50, 1993.

[16] M. Shaw, "Some Patterns for Software Architectures," Pattern Languages of Program Design Vol.2, pp.255-269, 1996.

[17] W. Keller, "Object/Relational Access Layers-A Roadmap, Missing Links and More Patterns." Proc. EuroPLoP 1998 Conf, July 1998.
[18] M. Adams, J. Coplien, R. Gamoke, R Hanmer, F. Keeve, and K. Nicodemus," Fault-Tolerant Telecommunication System Patterns," Proc. Secon Conf, Pattern Language of Program, Sept. 1995.
[19] Uwe Zdun, Markus Völter, Michael Kircher: Pattern-Based Design of an Asynchronous Invocation Framework for Web Services. Int. J. Web Service Res. 1(3): 42-62 (2004).
[20] Uwe Zdun, Markus Völter, Michael Kircher: Design and Implementation of an Asynchronous Invocation Framework for Web Services. ICWS-Europe 2003: 64-78.
[21] Nierstrasz Oscar, Meijler Theo Dirk, "Research Directions in Software Composition," ACM Computing Surveys, Vol.27, No.2, pp.262-264, June, 1995.
[22] Jim Q, Ning, "Component-Based Software Engineering," IEEE Software, 1997.
[23] Jan Bosch, Superimposition: A Component Adaptation Technique, Information and Software Technology, 41(5): 257-272, March, 1999.
[24] Urs Holzle. "Integration Independently-Developed Components In Object-Oriented Languages," Proceedings of ECOOP'93, Springer Verlag LNCS 512, 1993.
[25] John Cheesman, and John Daniels. "UML Components," Pearson Education: Addison-Wesley, 2000.

이 용 환
1997년 건국대학교 행정학사. 1999년 건국대학교 공학석사. 2006년 건국대학교 공학박사. 2005년 현재 (주)인터넷 커머스 코리아 연구소장과 동덕여대 겸임교수로 재임

민 덕 기
1986년 고려대학교 산업공학과 졸업 1991년 Michigan State 석사. 1995년 Michigan State 박사. 1995년 건국대학교 교수로 부임. 2005년 현재 건국대학교 부교수로 재임