가변적인 컴포넌트 개발을 위한 컴파일러 방식의 룰 엔진

(A Compiler Based Rule Engine for Developing Changeable Component)

이 용 환 †

(Yong Hwan Lee)

요 약 가변적인 컴포넌트의 재사용성이나 적응성을 높이기 위해 룰 기반 컴포넌트 개발 방법들이 제안되고 있다. 룰 기반 컴포넌트 개발에서 사용하는 룰 엔진들은 룰을 표현하기 위해 추가적인 스크립트 언어가 필요하며 따라서 복잡한 비즈니스 룰을 표현하는데 어려움이 많다. 본 논문에서는 다양한 룰 표현과 성능 향상을 위한 컴파일러 기반의 룰 엔진을 제안한다. 제안한 룰 엔진은 룰의 컨디션과 액션 부분을 표현하기 위해 자바 프로그래밍 언어를 사용한다. 따라서 복잡한 비즈니스 룰을 쉽게 표현할 수 있으며 실행 시에 동적으로 룰의 컨디션과 액션 객체를 생성해서 실행시킬 수 있다. 성능 면에서도 제안한 룰 엔진은 스크립트 기반 룰 엔진보다 우수하다. 성능 실험에 의하면 컴파일러 기반의 룰 엔진 성능은 스크립트 기반 룰 엔진인 JSR-94 보다 2.5배의 높은 성능을 보이고 있다.

키워드 : 룰, 가변적인 컴포넌트, 컴포넌트 적응성, 룰엔진, 룰기반 컴포넌트

Abstract To improve reusability and adaptation of variable components, rule-based component development has been used. Rule engines usually need additional script languages for rule expression and have difficulty in expressing complex business rules. In this paper, we propose a compiler-based rule engine for rich rule expression and improving performance. This rule engine uses Java programming language to express conditions and action parts of rules and that it can easily express complex business rules. It creates and executes condition and action objects at run time. In view of Performance, the rule engine is better than a script based rule engine. According to our experiments, our compiler-based rule engine shows 2.5 times better performance that script-based JSR 94 rule engine.

Key words : Rule, Variable Component, Component Adaptability, Rule Engine, Rule Based Component

1. 서 론

소프트웨어 산업이 급속하고 다양한 형태로 발전해가고 기업간 경쟁이 더욱 심화되면서 소프트웨어 재사용성, 적시성, 그리고 유지보수성이 중요시 되면서 컴포넌트 기반 소프트웨어 기술들이 점차 각광을 받고 있다. 컴포넌트 기반 개발에서는 컴포넌트의 재사용성도 중요하지만 새로운 요구사항이나 개발 당시와 상이한 요구사항에 적응하기 위한 컴포넌트의 적응성이나 확장성도 중요하다[1].

컴포넌트의 재사용 과정에서 발생하는 인터페이스 불일치, 구현 방식의 문제와 같은 컴포넌트 적응성 향상을 위해 래퍼(Wrapper)기법이나 BCA(Binary Component Adaptation)기법들이 제안되었다[2,3]. 이런 기법들은 언어에 의한 구현상의 차이로 인해서 한계는 존재하지만 모든 타입의 컴포넌트에 적용할 수 있다는 측면에서 높은 적용 가능성을 가지고 있다. 하지만 이러한 유연성은 성능 저하와 같은 많은 제한 점을 가져오고 있다.

가변적인 컴포넌트 자체의 적응성이나 재사용성을 높이기 위해서는 설계 단계에서 표현된 가변적인 부분들을 컴포넌트 내부의 코드와 분리해서 룰로 표현하고 룰 엔진을 사용해 실행해야 한다[4]. 하지만 스크립트 기반의 룰 엔진은 룰 표현을 위해 별도의 스크립트 언어가 필요하며 복잡한 룰 표현이 어렵다. 또한 스크립트 형태로 작성된 룰을 파싱하기 위해 인터프리터(Interpreter)

† 정 회 원 : 건국대학교 컴퓨터공학과 연구교수
yhlee@konkuk.ac.kr
논문접수 : 2006년 2월 16일
심사완료 : 2006년 8월 11일

방식을 사용하기 때문에 높은 성능을 요구하는 시스템에 적용하기에 어렵다는 단점도 있다. 본 논문에서는 다양한 룰 표현과 성능 향상을 위한 컴파일러 방식의 룰 엔진 설계 및 구현에 대해 기술한다.

본 논문에서 제안한 컴파일러 방식의 룰 엔진은 룰의 컨디션과 액션 부분을 자바 언어를 사용해 표현한다. 또한 자바 컴파일러를 사용해서 룰의 컨디션과 액션코드를 동적으로 클래스화 하거나 객체화해서 실행할 수 있다. 제안한 룰 엔진의 장점은 먼저 룰 표현을 위해 별도의 스크립트 언어가 필요하지 않는다. 둘째로 조건과 액션 코드가 인터프리터 방식이 아닌 자바 바이트 코드로 미리 컴파일 되어 있는 컴파일러 방식으로 작동하기 때문에 기존의 인터프리터 방식의 룰 엔진보다 성능 면에서 우수하다. 실험결과에 의하면 JSR-94보다 2.4배 성능이 우수하다. 세 번째로 조건이나 액션 코드 내부에서 기존 라이브러리를 사용 할 수 있기 때문에 보다 복잡한 조건문과 액션을 처리 할 수 있다. 즉, 다양한 문자열 처리, 숫자 처리, 논리 처리가 가능하다. 네 번째로 조건이나 액션코드가 실행 가능한 자바로 작성되어 있기 때문에 자바로 작성된 기존 시스템과 통합이 용이하다는 장점도 있다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 컴포넌트의 적응성을 위한 기법들을 소개하며 스크립트 기반의 룰 엔진인 JSR 94룰 엔진에 대해서 기술한다. 3장에서는 본 논문에서 제안한 컴파일러 방식의 룰 엔진 아키텍처에 대해 기술하고 4장에서는 스크립트 기반 룰 엔진인 JSR-94와 비교한다. 5장에서는 기존에 비즈니스 컴포넌트 개발을 위해 제시했던 룰 기반 분석 패턴[5]을 통해 식별된 컴포넌트들에 룰 엔진을 적용해서 컴포넌트를 개발하는 방법을 기술한다. 마지막으로 6장에서는 결론을 기술한다.

2. 관련연구

컴포넌트는 크게 비즈니스 컴포넌트와 어플리케이션 컴포넌트로 분류된다. 비즈니스 도메인의 공통 컴포넌트 재사용 측면에서 비즈니스 컴포넌트의 의미는 크다. 이러한 비즈니스 컴포넌트는 비즈니스 프로세스 컴포넌트, 비즈니스 도메인 컴포넌트, 그리고 데이타 컴포넌트로 분류된다. 비즈니스 프로세스 컴포넌트는 비즈니스 트랜잭션을 처리하고 하나의 비즈니스 업무의 처리 요구사항들이나 업무를 처리하기 위한 운영상의 흐름을 정의하고 있다. 비즈니스 도메인 컴포넌트는 비즈니스 단위의 도메인 고유의 서비스를 제공한다. 데이타 컴포넌트는 비즈니스 데이타 처리와 관련된 서비스를 제공한다.

컴포넌트 기반 소프트웨어 재사용 기법은 적응, 수정, 그리고 조립으로 분류된다[6-8]. 수정 기법은 컴포넌트 재 사용이나 적응성 향상을 위해 컴포넌트 속성들(Properties)을 변경한다. 컴포넌트의 속성은 일반적으로 컴포넌트의 행위나 범위를 결정짓는 피쳐들(Features)이다. 조립기법에서는 컴포넌트 재사용을 위해 조합이나 통합과 같은 기법을 사용한다[9]. 적응 기법은 이미 정의된 인터페이스나 컴포넌트 행위를 변경하기 위해서 사용하는 것으로서 적응 기법은 개조기(adapter)기법, Wrapper 기법, Superimposition 기법, 그리고 Binary Adaptation기법[10] 등이 존재한다. 이러한 적응 기법들은 언어에 의한 구현상의 차이로 인한 한계는 존재하지만 모든 타입의 컴포넌트에 적용할 수 있다는 측면에서 높은 적용 가능성을 가지고 있다. 하지만 이러한 유연성은 성능 저하와 같은 많은 제한 점을 가져오고 있다. 이러한 문제점을 해결하고 컴포넌트 적응성이나 재 사용성을 향상시키기 위해 최근에는 룰 기반 컴포넌트 개발 방법들이 제안되었다[11].

JSR 94는 자바 기반 룰 엔진 구현을 위한 표준을 제공하기 위해 썬 마이크로 시스템에서 제공한 룰 엔진이다. JSR 94는 룰을 표현, 파싱 그리고 실행 하기 위해 내부적으로 Jess 룰 엔진[12]을 사용하며 그 기반 위에 룰을 등록, 제거, 필터적용. 세션관리와 같은 추가적인 기능들을 구현하고 있다[13]. JSR 94는 룰을 표현하기 위해 CLIPS 전문가 시스템 쉘(CLIPS expert system shell)로부터 진화된 Jess 룰 스크립트 언어를 사용한다. JSR 94를 사용해 비즈니스 룰을 표현하기 위해서는 복잡한 Jess 룰 스크립트 언어를 배워야 하기 때문에 가변적인 비즈니스 로직이 복잡한 경우에 현실적으로 적용하는데 어려움이 있다. 또한 룰 실행 시 인터프리터 방식으로 룰 스크립트를 파싱해야 하기 때문에 본 논문에서 제안하는 컴파일러 방식의 룰 엔진보다 성능이 떨어진다. 이에 본 논문에서는 룰 표현이 용이하고 JSR 94보다 성능 면에서 우수한 컴파일러 기반 룰 엔진을 제안한다.

3. 컴파일러 기반 룰 엔진 설계

3장에서는 컴파일러 방식의 룰 엔진의 내부 아키텍처와 룰 엔진 초기화와 룰 실행 요청에 대한 처리흐름에 대해 기술한다.

3.1 룰엔진 아키텍처

그림 1은 본 논문에서 제안한 컴파일러 방식의 룰 엔진 아키텍처이다. 룰 엔진은 크게 어드민 콘솔, XML기반 룰 저장소 그리고 룰 엔진 시스템으로 구성된다. 먼저 어드민 콘솔은 룰을 표현하고 관리하기 위한 툴킷이며 XML기반 룰 저장소는 어드민 콘솔에서 표현된 룰을 XML형태로 저장한다. 마지막으로 룰 엔진 시스템은 룰을 실행하기 위한 엔진 부분이다.

Rule Engine역할은 클라이언트로부터 룰 처리 요청 메시지를 받아 적절한 룰을 실행시키는 것이다. 적절한 룰을 찾기 위해 도착된 요청 메시지를 Rule Parser에게 전송한다. Rule Parser는 요청 메시지로부터 이벤트 식별자를 추출해서 파싱 테이블의 룰 식별자와 비교해 적절한 룰을 찾는다. 이벤트 식별자와 룰 식별자는 "도메인 이름: 업무 구분자: 룰 이름" 형태로 되어 있는 문자열이다. 해당 룰을 찾은 후에 룰 엔진은 파싱테이블에서 컨디션과 액션 부분에 대한 객체 이름을 가져와서 ObjectPool Manager를 통해 해당 객체의 참조 값을 얻어 룰을 실행 한다.

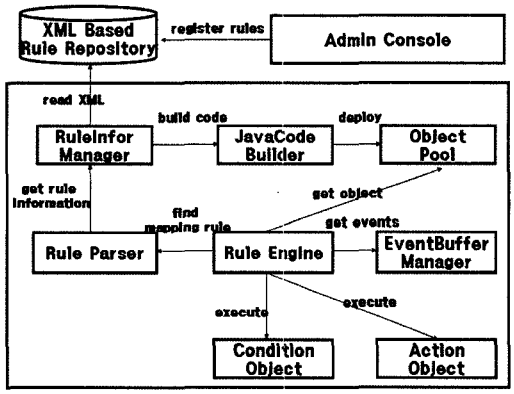

그림 1 컴파일러 방식의 룰엔진 아키텍처

Rule Parser는 Rule Engine으로부터 요청을 받아 해당 룰을 찾아주는 역할을 하며 ObjectPool Manager는 특정 룰에 표현된 컨디션 객체와 룰 객체들을 관리한다. RuleInfor Mananger는 XML기반 룰 저장소에 관련된 CRUD(Create, Read, Update, Delete)을 처리한다. JavaCode Builder는 자바 언어를 사용해 정의된 룰의 컨디션과 액션 부분을 자바 컴파일러를 통해 클래스 바이트 코드를 생성하고 객체화 한 다음에 객체 풀에 배포한다. Condition과 Action객체는 자바 언어를 통해서 표현된 룰의 컨디션과 액션에 관련된 객체들이다.

룰 엔진은 룰 실행 전에 초기화를 수행해야 한다. 그림 2는 룰 엔진 초기화를 수행하는 처리과정을 보여주는 Collaboration Diagram이다.

초기화 요청을 받은 Rule Engine은 먼저 RuleInfor Manager에게 초기화 요청을 보낸다. RuleInfor Manager는 XML저장소에 저장되어 있는 모든 룰 표현 정보를 읽어 버퍼에 저장한다. RuleInfor Manager는 반복적으로 룰 표현에 있는 각 컨디션과 액션부분에 정의되어 있는 자바 코드를 JavaCode Builder를 사용해 자바 소스코드를 생성하고 자바 컴파일러를 사용해 클래스 바

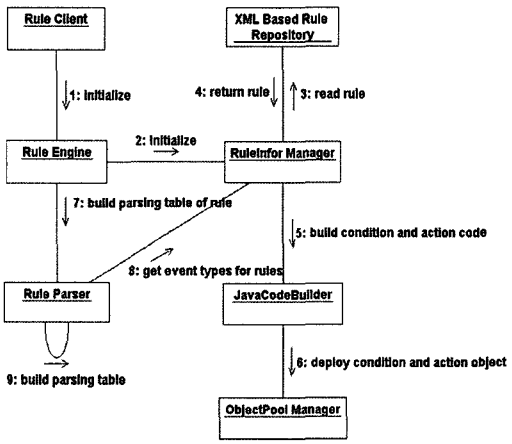

그림 2 룰 엔진 초기화

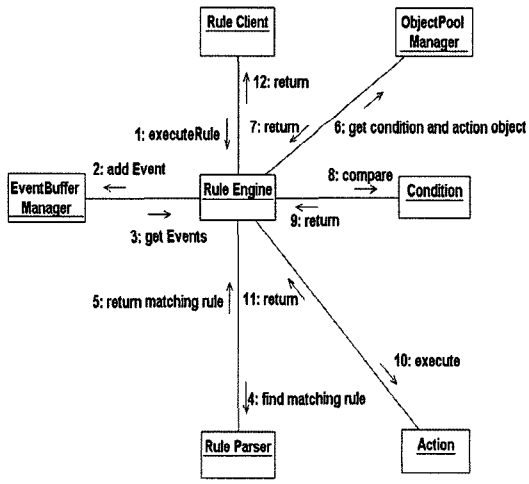

그림 3 룰 엔진의 룰 실행 처리 과정

이트 코드와 그에 대한 객체를 생성해서 ObjectPool Manger를 통해 객체 풀에 저장한다. 각 룰에 있는 컨디션과 액션 객체를 초기화 한 다음에 Rule Engine은 파싱 테이블을 구축하기 위해 Rule Parser를 호출한다. Rule Parser는 RuleInfor Manager를 통해 룰 식별자와 그 룰에 대한 정보들을 가져와 파싱 테이블을 구성한다.

그림 3은 룰 클라이언트로부터 룰 실행 요청을 받았을 때 룰 엔진에서 룰을 실행하는 처리과정을 보여주는 Collaboration Diagram이다.

룰 클라이언트는 룰 엔진에게 룰 처리 요청 메시지를 전송한다. 룰 엔진은 먼저 EventBuffer Manager를 호출해서 룰 처리 요청 메시지를 버퍼에 저장한다. 룰 엔진은 버퍼에 저장되어 있는 요청 메시지 중에서 우선순위가 높은 메시지부터 처리한다. 다음으로 Event

Engine은 요청 메시지 내부에 있는 이벤트 식별자와 매칭되는 룰을 찾기 위해 Rule Parser를 호출한다. Rule Parser는 파싱테이블을 이용해 해당 룰을 찾기 위해 이벤트 식별자와 파싱테이블에 있는 룰 식별자를 비교하여 해당 룰을 찾는다. 해당 룰을 찾은 Rule Engine은 ObjectPool Manager를 호출해서 해당 룰에 표현되어 있는 컨디션과 액션 객체를 얻는다. 그런 다음 컨디션 객체 내부에 있는 compare라는 훅 메소드를 호출한다. 훅 메소드 호출 결과가 참이면 룰에 표현된 액션 객체의 훅 메소드 execute를 실행 시킨다. 만일 룰에 표현된 액션 객체가 여러 개인 경우 룰에 표현된 액션 객체 순서대로 실행 시킨다. 즉, 본 논문에서 제안한 룰 엔진은 Forward-Chaining방식의 룰 실행을 지원한다. 따라서 어떤 룰의 액션 실행 결과가 다른 룰의 컨디션 입력값으로 사용된다.

4. JSR-94룰 엔진과의 비교

본 장에서는 룰 표현과 성능 관점에서 JSR-94룰 엔진과 비교한다. 두개의 룰 엔진을 비교하기 위해 수출입 업무와 관련된 고객 신용한도 관리 룰을 가정하자. 만일 고객의 신용 한도 금액이 송장의 금액보다 크고 동시에 송장의 지불 상태가 지불되지 않는 상태일 경우 고객의 신용 한도를 송장의 금액만큼 감소시키고 동시에 송장의 상태를 지불 상태로 변경하는 비즈니스 룰이 있다고 하자. 그림 4는 앞에서 설명한 고객 신용한도 관리 룰을 JSR-94에서 사용하는 Jess 룰 스크립트 언어를 사용해 표현하는 예제를 보여주고 있다.

```
<rule-execution-set>
 <name>RuleExecutionSet1</name>
 <description>Stateless RuleExecutionSet for the TCK for JESS</description>
 <code>
  (defclass org.jcp.jsr94.tck.model.Customer
        org.jsp.jsr94.tck.model.Customer)
  (defclass org.jcp.jsr94.tck.model.Invoice
        org.jsp.jsr94.tck.model.Invoice)
  (defrule rule-1
        ?customer <- (org.jcp.jsr94.tck.model.Customer
                     (creditLimit ?limit) (OBJECT ?C)
        ?invoice  <- (org.jcp.jsr94.tck.model.Invoice
                     (amount ?amt&:(> ?limit ?amt))
                     (status "unpaid") (OBJECT ?I))
        =>
        (modify ?customer (creditLimit (- ?limit ?amt)))
        (printout t "The credit limit of the customer is "
                     (get ?C creditLimit) crlf)
        (modify ?invoice (status paid))
        (printout t "The status of the invoice is "
                     (get ?I status) crlf)
 </code>
</rule-execution-set>
```

그림 4 Rule Expression Using Jess Rule Script Language

JSR-94는 비즈니스 룰을 표현하기 위해 별도의 복잡한 스크립트 언어를 배워야 하며 룰을 파싱 할 때 인터프리터 방식을 사용하기 때문에 빠른 성능을 요구하는 시스템에 적합하지 않는다.

그림 5는 똑같은 고객 신용한도 관리 룰을 본 논문에서 제안한 컴파일러 기반 룰 엔진에서 사용하는 관리 콘솔을 사용해서 룰을 정의하는 방식이다.

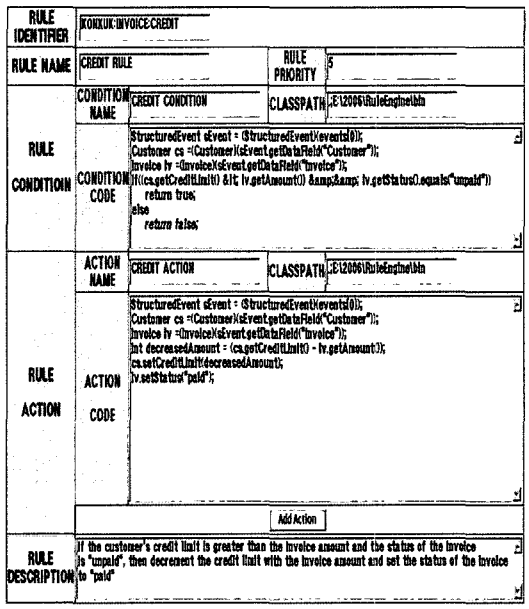

그림 5 관리 콘솔을 사용한 룰 표현

관리 콘솔은 트리형태로 업무를 구분하고 있으며 룰 정의는 각 업무 단위 별로 정의할 수 있다. 만일 같은 업무 단위에 룰이 여러 개인 경우 고유한 룰 이름을 사용해서 구분한다. 룰의 기본 구문인 컨디션과 액션에 해당하는 부분들이 자바 언어를 사용해서 표현하고 있으며 또한 룰의 컨디션이나 액션부분 실행을 위한 CLASSPATH형태로 라이브러리를 지정해서 사용할 수 있기 때문에 다양한 문자열 처리, 숫자 처리, 논리 처리가 가능하다. 룰의 액션 부분과 컨디션 코드 부분은 각각 컨디션과 액션 자바 객체의 훅 메소드 내부에 포함되며 자바 컴파일러를 사용해서 컴파일 되며 그 클래스의 객체가 객체 풀에 배포된다. 그림 5의 Rule Identifier는 적절한 룰을 찾기 위해 사용되며 Rule Priority는 룰을 실행하는 순서를 지정한다.

JSR-94는 Interpreter에 의해 룰들을 파싱 한다. 따라서 JSR-94는 높은 성능을 요구하는 시스템에는 적합하지 않다. 하지만 본 논문에서 제안한 룰 엔진은 룰에 지정된 컨디션과 액션 객체를 객체 풀에서 가져와서 실

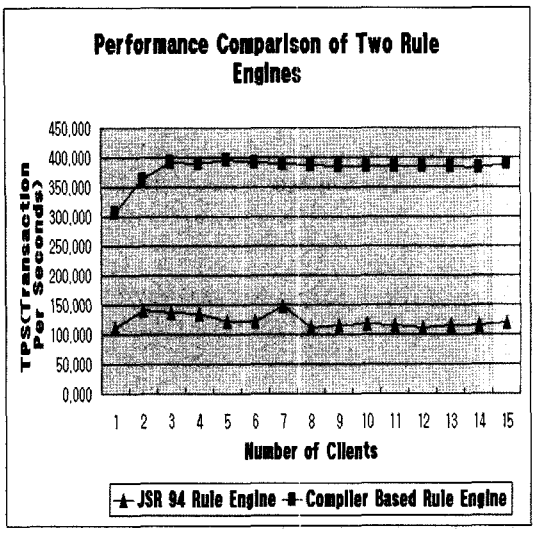

그림 6 JSR 94룰 엔진과 컴파일러 기반 룰 엔진 성능 비교

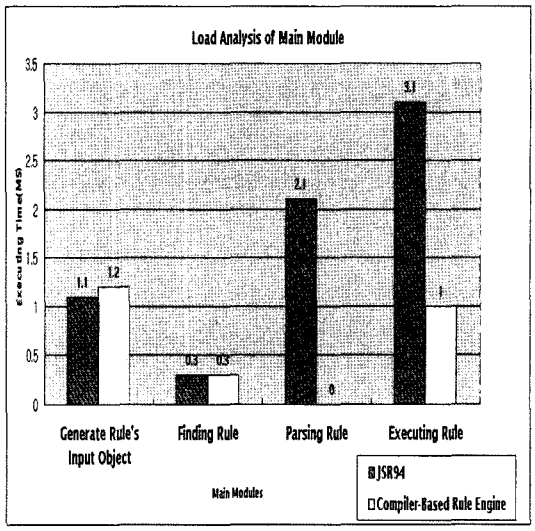

그림 7 두 개의 룰 엔진 주요 모듈 비교

행한다. 따라서 성능측면에서 JSR-94보다 우수한다. 그림 6은 본 논문에서 제안한 룰 엔진과 JSR-94와의 성능비교 결과를 보여주고 있다.

성능 실험을 위한 운영 시스템으로 Microsoft 2003 서버를 사용했고 어플리케이션 서버를 위해 WebLogic 6.1 with SP 7를 사용했다. 워크로드 생성을 위해 WebBench 5.0을 사용했으며 성능 메트릭(Metrics)을 위해 TPS(Transaction Per Seconds)를 사용하며 워크로드를 위해 앞에서 설명한 고객 신용 한도 관리를 사용한다. 또한 J2EE환경에서 성능 비교 결과를 보기 위해 룰 엔진의 클라이언트로서 간단한 서블릿을 사용했다.

룰 엔진은 최대 395 TPS를 보이고 있으며 반면에 JSR-94는 150 TPS를 보이고 있다. 본 논문에서 제시한 룰 엔진이 JSR-94보다 초당 245개의 트랜잭션을 더 처리할 수 있으며 거의 2.4배 정도의 성능이 우수하다. 그림 7은 두 개의 룰 엔진의 주요 모듈의 성능을 비교한 결과이다. 본 논문에서 제시한 룰 엔진은 컴파일 기반으로 룰을 처리하기 때문에 룰과 관련된 컨디션과 액션 객체를 생성하는데 많은 시간을 소모하고 있지만 JSR-94와 비교해서 실행 시간이 더 걸리고 두개의 엔진이 룰을 찾는데 걸리는 시간은 동일하다. 하지만, 룰을 분석하는데 있어서는 본 논문에서 제안한 컴파일러 기반의 룰 엔진은 스크립트 형태로 되어 있지 않고 자바 실행 코드 형태로 실행되기 때문에 별도의 룰 분석 과정이 필요하지 않다. 또한, 룰을 실행하는데 있어서도 객체 풀에서 룰의 컨디션과 액션 객체를 사용하기 때문에 JSR-94보다 높은 성능을 보이고 있다.

JSR-94룰 엔진은 룰을 파싱 하는 부분에서 가장 심한 부하를 받고 있다. 하지만 본 논문에서 제안한 룰 엔진은 룰의 컨디션과 액션 부분들이 자바 코드로 되어 있어 있고 더욱이 컨디션과 액션 관련 객체들을 객체 풀에서 얻어 실행하기 때문에 파싱에 따른 부하를 줄일 수 있다. 또한 룰 엔진 실행 시에 룰의 컨디션과 액션 부분을 변경해서 객체 풀에 배포하기 때문에 동적으로 룰 변경이 가능하다는 장점도 가지고 있다. 더욱이 룰의 컨디션과 액션 부분이 스크립트가 아닌 자바 코드로 되어 있기 때문에 기존 시스템과 통합이 용이하며 또한 자바로 작성된 시스템을 룰 기반으로 변형하기가 용이하다는 장점도 있다.

5. 컴포넌트 개발에 룰엔진 적용 방법

룰 기반 컴포넌트 개발 접근 방법은 컴포넌트의 가변적인 부분들을 별도의 룰 형태로 분리하고 룰 실행을 위해 룰 엔진을 사용한다. 룰 엔진을 컴포넌트 개발에 적용하기 위해서는 두개의 대상 컴포넌트가 존재한다.

첫 번째 대상 컴포넌트는 개별적인 비즈니스 도메인 컴포넌트이다. 비즈니스 도메인 컴포넌트는 개별적인 업무 단위의 서비스를 제공하는 컴포넌트이다. 룰 엔진을 개별적인 비즈니스 도메인 컴포넌트에 적용함으로서 비즈니스 도메인 컴포넌트의 재사용성과 적응성을 달성할 수 있다. 첫 번째 접근 방식은 비즈니스 도메인 컴포넌트로부터 가변적인 비즈니스 로직을 분리하고 그것들을 룰로 표현해서 실행하는 방식이다. 비즈니스 도메인 컴포넌트에서 사용하는 룰은 일반적으로 대상이 가지고 있는 가변적인 특성과 특성을 해석하는 가변적인 방법

을 정의한다. 예를 들면 고객의 나이를 구하는 도메인 서비스는 문제의 특성에 따라서 은행에서 사용하는 나이일 수도 있고, 주민번호에 의한 법적 나이일수도 있다. 이는 동일한 컴포넌트가 어떤 비즈니스 영역에 적용되느냐에 따라 적용할 비즈니스 룰이 달라지는 경우이다.

두 번째 대상 컴포넌트는 개별적인 컴포넌트 상호작용에 의해 서비스를 제공하는 비즈니스 프로세스 컴포넌트이다. 비즈니스 프로세스 컴포넌트는 비즈니스 트랜잭션을 처리하며 비즈니스 서비스 단위 별 처리흐름이나 요청 요구사항을 정의한다. 비즈니스 프로세스 컴포넌트에서 사용하는 룰은 하나의 업무를 처리하는데 필요한 작업종류, 순서, 그리고 처리 조건 등을 정의 하는 룰이다. 비즈니스 프로세스 컴포넌트에서 이러한 가변적인 컨트롤 로직을 분리해 별도의 룰로 정의하고 실행하는 방식이다.

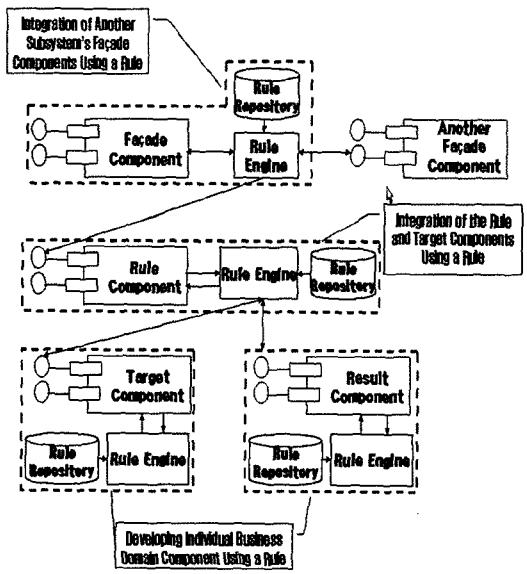

그림 8 분석 패턴에 의해 식별된 컴포넌트 개발에 룰 엔진 적용

최근에 우리는 비즈니스 컴포넌트 개발을 위해 룰 기반 분석 패턴을 제시했다[5]. Façade, Rule, Target, 그리고 Result는 기존에 제시했던 룰 기반 분석 패턴을 통해 식별된 컴포넌트 타입들이다. Facade컴포넌트는 두 가지 역할을 가지고 있다. 첫 번째는 외부 시스템에 대한 파사드(Façade) 역할을 수행하는 것이고 다른 역할은 다른 서브 시스템에 존재하는 Facade컴포넌트와의 상호 작용과 같은 비즈니스 워크플로우 역할이다. 컴포넌트의 변경 가능성이나 재사용성을 용이하게 하기 위해 다른 서브 시스템상의 Facade컴포넌트와의 상호 작용은 Rule, Target, 그리고 Result가 아닌 Facade컴포넌트를 통해 수행된다. 룰 컴포넌트는Target과 Result 컴포넌트 사이의 상호작용을 조절하기 위한 비즈니스 프로세스 컴포넌트이다. 하지만 다른 서브시스템의 Facade 컴포넌트와의 상호작용은 조정하지 않는다. Target과 Result컴포넌트는 업무 구분 별 서비스를 제공하기 위한 비즈니스 도메인 컴포넌트들이다.

그림 8은 룰 기반 분석 패턴을 통해 식별된 컴포넌트들에 룰 엔진을 적용하는 접근 방식을 보여주고 있다. 룰 엔진을 Façade나 Rule컴포넌트에 적용하는 것은 다른 컴포넌트와의 상호작용에 대한 적응성을 향상시키기 위해서이다. Facade컴포넌트 경우에는 룰 엔진은 다른 서브시스템상의 Facade컴포넌트와 상호작용에 대한 적응성을 향상 시키기 위함이며 Rule컴포넌트에 룰 엔진을 적용하는 경우는 Target과 Result와의 상호작용의 적응성을 높이기 위함이다. 하지만 Target과 Result컴포넌트는 개별적인 비즈니스 도메인 컴포넌트의 적응성 향상을 위해 룰 엔진을 사용한다.

6. 결론 및 향후 연구

컴포넌트를 설계할 때 가장 중요한 측면은 컴포넌트의 확장성과 적응성을 위해 컴포넌트 내부에서 가변성을 제거하는 것이다. 많은 연구자들은 컴포넌트 적응성 향상을 위한 많은 방법들을 제안했다. 하지만 유연성 향상이 성능을 저하시키는 예제와 같이 어떤 장점들이 다른 약점을 가져오는 한계가 존재한다. 이러한 문제점을 해결하기 위해 룰 기반 컴포넌트 개발 방법들이 제안되고 있다. 하지만 이들 룰 기반 기법들은 일반적으로 룰을 파싱 하기 위해 인터프리터 기반의 룰 엔진을 사용하고 있다. 이들 방식들은 추가적인 룰 스크립트 언어가 필요하며 높은 성능을 요구하는 시스템에 적합하지 않다. 본 논문에서는 복잡한 룰 표현과 성능을 향상시키기 위해 컴파일러 기반의 룰 엔진을 제안한다. 제안한 룰 엔진은 룰의 컨디션과 액션 부분을 자바 언어를 사용해서 표현하고 있다. 따라서 복잡한 비즈니스 로직을 자바 프로그래밍과 유사하게 표현할 수 있으며 또한 자바 언어에서 사용하는 다양한 라이브러리를 그대로 사용할 수 있기 때문에 복잡한 문자, 숫자, 그리고 논리 표현 등을 쉽게 할 수 있다. JavaCode Builder를 사용해 룰의 컨디션과 액션 부분 자바 코드를 자바 객체로 생성해서 객체 풀에 배포하기 때문에 룰 엔진 실행 시에 동적으로 룰을 변경할 수 있다. 또한 인터프리터 방식이 아닌 컴파일러 방식으로 룰을 파싱해서 실행하고 객체 풀을 사용하기 때문에 다른 스크립트 기반 룰 엔진보다 성능이 우수하다. 실험 결과에 의하면 JSR-94보다 무려 2.4배나 우수하다.

본 논문에서 제안한 컴파일러 기반 룰 엔진은 상태가 유지되지 않는 Stateless기반의 동기식 방식이다. 향후에는 Stateful기반의 비 동기식 룰 처리를 고려할 것이다.

참 고 문 헌

[1] M.Morisio and C.B.Seaman et al, "Investigating and improving a COTS-based software development process," ICSE 2000, pp. 31-40, 2000.

[2] Jim Q. Ning, "Component-Based Software Engineering," IEEE Software, 1997.

[3] Jan Bosch. "Superimposition: A Component Adaptation Technique. Information and Software Technology," 41(5):257-273, March 1999.

[4] Lars Geyer and Martin Becker, "On the influence of Varaibilities on the Application-Engineering Process of a Product Famliy," Proceedings of SPLC2, 2002.

[5] Yonghwan Lee, Eunmi Choi, Dugki Min, "A Rule Based Analysis Method for Cooperative Business Application," GCC 2005, LNCS 3795, pp. 1155-1160. 2005.

[6] Peter Herzum and Oliver Sims, "The Business Component Approach," OOPSLA'98 Business Object Workshop IV, 1998.

[7] Nierstrasz Oscar, Meijler Theo Dirk, "Research Directions in Software Composition," ACM Computing Surveys, Vol. 27, No. 2, pp. 262-264, June 1995.

[8] Johannes Sametinger, "Classification of Composition and Interoperation," OOPSLA'96 Poster Presentation.

[9] Nierstrasz Oscar, Meijler Theo Dirk, "Research Directions in Software Composition," ACM Computing Surveys, Vol. 27, No. 2, pp. 262-264, June 1995.

[10] Ralph Keller and Urs Hlzle, "Binary Component Adaptation," Lecture Notes in Computer Science, vol.1445, 1998.

[11] Jeong Ah Kim, JinYong Taek, SunMyung Hwang, "Rule-based Component Development," SERA 2005, Third ACIS International Conference on 11-13 Aug. 2005 Page(s):70-74.

[12] Jess Rule Engine, http://herzberg.ca.sandia.gov/jess/.

[13] JSR-94, http://www.jcp.org/aboutJava/communityprocess/reviewer/jsr094.

[14] Peter Herzum and Oliver Sims, "Business Component Factory," OMG Press, 2000.

이 용 환

정보과학회논문지 : 컴퓨팅의 실제

제 12 권 제 2 호 참조