# Prolog/XML를 이용한 비즈니스 룰(Business Rule) 분석 시스템 설계

권순덕[O] 이원조 이단영 고재진
울산대학교 컴퓨터 · 정보통신 공학부
{trueguy[O]}@cic.ulsan.ac.kr

## Using Prolog/XML for business rules implementation system design

Sun-Deok Kwon[O] Dan-Young Lee Won-Jo Lee Jea-Jin Ko
School. of Computer Engineering · Information Technology. Ulsan University

**요 약**

본 논문에서는 EDI(Electronic Data Interchange)와 WorkFlow 시스템에서 가지고 있는 비즈니스 룰(Business Rule)을 논리 프로그램인 prolog를 이용해서 각 비즈니스 룰을 분석하고 각각의 프로세스를 완전한 수행을 할 수 있는 시스템을 설계한다.

## 1. 서 론

### 1.1 비즈니스 프로세스(Business Process)

비즈니스 프로세스는 전통적인 기업방침을 사용하지 않고 기업 전체를 대상으로 하는 기간 시스템으로서 기업 내의 모든 업무와 자원을 프로세스 관점에서 통합하여 관리하게 된다. WFMS(WorkFlow Management System) 의 통합 대상으로는 기업 전체의 인적, 물적 자원에 대한 관리 모델이 포함된다. 하지만 기존에 구축되어 시스템을 배제하고 새로운 시스템을 구축한다면 완전히 새로운 시스템을 구축해야 한다는 부담과 경제적인 부담이 가중될 것이다. 따라서 기존의 정보 시스템과 WFMS를 통합하려는 노력이 시도되고 있다. 특히, WFMS를 현존하는 기업 내부 데이터와 연계시키고, 프로세스 데이터를 안정된 트랜잭션의 관점 상에서 다루기 위하여 데이터베이스 상에서 WFMS를 구축하는 연구가 시도되고 있다. 이 때 조직 내의 존재하는 정보 시스템은 워크플로우 시스템이 수행해야 할 프로세스의 요구 사항(데이터의 무결성, 정보 자원의 적합성 등)을 충분히 반영함으로써 프로세스의 목적을 달성할 수 있게 된다. 비즈니스 프로세스 리엔지니어링이 프로세스 그 자체를 대상으로 하여 조직 구조와 업무 전반의 근본적인 개혁을 시도하는 것이라면, WFMS는 변화된 프로세스가 정보 시스템 상에서 올바르게 수행되도록 하는 도구의 역할을 한다. 또한 WFMS는 업무를 수행하기 위해서 각 업무의 비즈니스 룰을 가지고 있다. 올바른 작업을 수행하기 위해서 항상 룰 기반의 작업을 수행해야 한다.

### 1.2 Prolog

논리지향적 언어이다. 논리지향적 언어이다. 특히 술어논리(Predicate Logic)에 기반을 두고 있다. PROLOG는 대상(Object)과 대상간의 관계(Relationship)를 포함하고 있는 문제를 해결하는데 사용되는 언어이다. 예를 들어, "John은 책을 가지고 있다."는 문장은 John이라는 대상과 책이라는 또 다른 대상간의 소유관계를 선언하고 있다(관계에서는 대상간의 순서가 중요). PROLOG 프로그래밍은 다음과 같은 사항들로 되어 있다.

1) 대상과 그들간의 관계에 대한 사실(Facts) 선언
2) 대상과 그들간의 관계에 대한 규칙(Rules)의 정의
3) 대상과 그들간의 관계에 대한 물음(Questions)

따라서 PROLOG 시스템은 사실들과 규칙들로 이루어져 있고, 이것들로 물음에 대한 대답을 한다.

#### 1.2.1 Prolog의 기본 구성요소

①사실(Facts)

"John은 Mary를 좋아한다"는 사실은 PROLOG에서는 다음과 같이 표현될 수 있다.

```
likes(john, mary).
```

여기서 john과 mary는 대상이고, likes는 관계이다.
사실들로 대상간의 관계를 정의할 때에는 괄호 안에 열거될 대상들의 순서에 주의하여야 한다. 위의 예에서 likes(mary, john)과 같이 john과 mary의 순서를 바꾸면 이는 "Mary는 John을 좋아한다"로 해석된다.

물음(Questions)
어떤 사실이 있다면 그것에 대한 물음을 할 수 있다. PROLOG에서는 물음은 사실(Facts)과 비슷하게 표현하지만, 그 앞에 특수한 심볼을 붙여야 한다. 이 심볼은 물음표와 하이픈으로 구성된다. PROLOG로 표현된 다음과 같은 질문을 생각하여 보자.

?- owns(mary, book).

이 물음에서 mary를 "Mary"라 불리우는 사람으로 해석을 하고, book을 어떤 특정의 책을 의미한다고 하면, 이 물음은 "Mary가 그 책을 가지고 있는가?"라는 물음이 된다.

③변수(Variables)
"John은 Mary를 좋아합니까?"라고 묻는 대신에 "John은 X를 좋아합니까?"라고 묻는다면 X가 무엇을 의미하는지 알 수가 없다. 이 경우 X를 변수라 부른다. PROLOG에서 변수를 사용하는 경우, 변수는 instantiate되거나 not instantiate되거나 한다. instantiate가 되었다는 것은 변수가 뜻하는 대상이 존재한다는 의미이다. 변수는 알파벳 대문자로 시작함으로써 대상들의 명칭과 구별된다.
다음과 같은 사실들로 구성된 데이터베이스가 있다고 하자.

likes(john, flowers).
likes(john, mary).
likes(paul, mary).

이것들에 대하여 다음과 같은 물음이 있다고 하자.

?- likes(john, X).

이 물음은 "John이 좋아하는 것이 무엇입니까?"라는 의미이며, 이에 대하여 PROLOG는 X=flowers라고 답한다.

④논리적(Conjunctions)
데이터베이스에는 John은 Mary를 좋아한다는 사실이 있으면 첫 번째 목표는 참이다. 그러나 Mary가 John을 좋아한다는 사실은 없다면 두 번째 목표는 거짓이다. 우리는 데이터베이스에 저장된 사실을 근거로 새로운 사실을 유추할 수 있다.

⑤규칙(Rules)
모든 사실을 일일이 열거하는 것 보다 규칙의 형태로 표현한다면 훨씬 효율적일 것이다. 규칙은 어떤 사실이 다른 사실들의 집합에 의존적일 경우에 사용된다. 이때 규칙을 표현하는데 만일(if)이라는 단어가 사용된다.
(예)
나는 만일 비가 오면 우산을 사용한다.
규칙은 또한 어떤 것을 정의하는데 사용된다.
(예)
만일 X가 동물이고 날개가 있다면 X는 새이다.(1)
만일 X와 Y가 여성이고 X와 Y의 부모가 같다면, X와 Y는 자매간이다.(2)

규칙은 대상과 그들간의 관계에 대한 일반적인 표현이다. 규칙은 head와 body로 구성되며, ":-"로 연결된다. 아래의 예는 규칙으로 다음과 같이 표현된다.
"John은 포도주를 좋아하는 사람을 좋아한다."
또는 변수를 사용하여
"만일 X가 포도주를 좋아한다면 John은 X를 좋아한다."

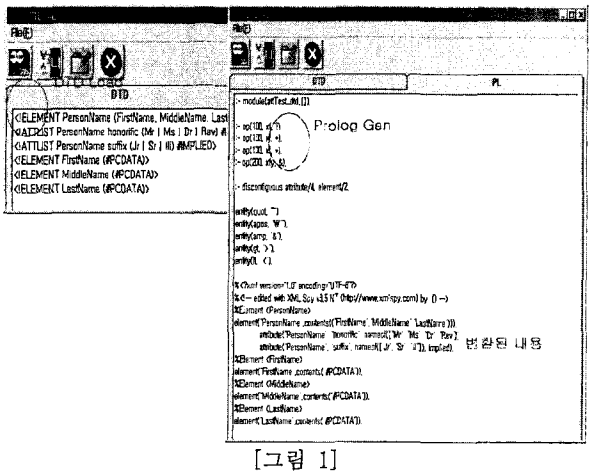
likes(john, X) :- likes(X, wine).

규칙의 head부분은 likes(john, X)이며 규칙이 정의하고 싶은 사실을 기술한다. 규칙의 body부분은 likes(X, wine)이며 head가 참이기 위해서 만족시켜야할 목표들을 기술한다.

## 2. 연구 내용

### 2.1 XML / Prolog

XML의 데이터 교환 포맷은 매우 융통성이 있다. XML이 없다면 두 통신용 애플리케이션은 그들 간에 전송될 데이터 요소 및 데이터 요소를 사전에 정의해야 한다. 그러나 XML이 메시지 포맷인 경우에는 두 애플리케이션은 XML구문분석기를 사용하여 능동적으로 메시지 포맷을 해석한다. [그림 1]에서는 XML문서를 Prolog로 전환하는 모습을 보여준다.




[그림 1]

XML는 기존의 비즈니스 룰에서 필요로 하는 값을 엘리먼트로 추출하여 XML DTD로 정의한다. 정의된 DTD를 이용해서 Prolog로 전환한다. 이 단계에서 Prolog는 비즈니스 룰을 인식하고 Prolog적 표현법으로 변환하다. 그리고 데이터베이스에 사실(fact)와 규칙(rules)가 저장된다.

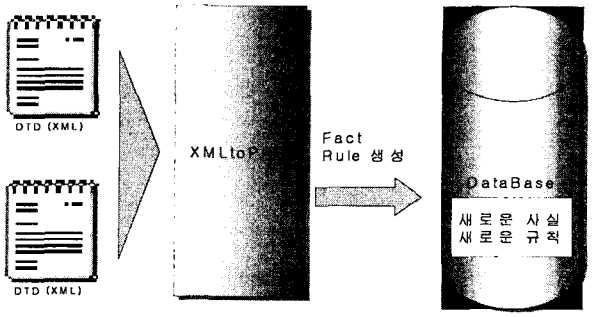


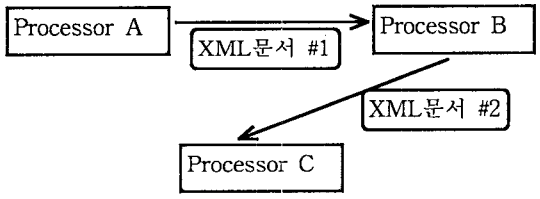
그림2]에선 DTD를 Prolog로 변환하는 단계를 보여준다.

| XML 표기: |
| --- |
| <!ATTLIST Element Attribute Type Default> |
| Prolog 표기: |
| attribute(Element, Attribute, Type, Default) |

[그림 2]

2.2 룰(Rule) 기반의 상호 작용하는 프로세스

하나의 완전한 작업을 수행하기 위해서는 서로 다른 프로세스간의 상호 작용이 필요할 경우가 있다. 하나의 프로세스의 결과가 또 다른 프로세스의 입력 값이 되는 경우가 이와 같은 경우일 것이다. 이런 상호 보완적인 여러 작업이 완전히 완료되었을 때 비로소 하나의 작업이 끝나게 된다. 하지만 이런 작업의 흐름에서 하나의 작업이라도 에러가 발생하면 이와 연관된 모든 작업의 결과는 이전의 상태로 돌아가야 한다. 즉 작업의 결과로 생성된 내용이 이전의 값으로 다시 복구해야 한다. 이런 규칙은 DTD에 정의해 놓고 Prolog가 분석한 후 룰(rules)과 규칙(facts)를 데이터베이스에 저장한다. 그리고 각 프로세스를 처리하기 전에 Prolog 엔진를 이용해서 각 프로세스가 참(true)일 경우만 수행한다.




[그림 3]

▶XML문서에서는 문서가 가지고 있는 고유의 Key를 가지고 있다. (각 문서를 식별하기 위한 Key)

▶서로 연관된 작업에서 작성된 XML문서는 작업에 의해서 생성된 값의 히스토리 기능을 가지고 있어야 한다. (작업의 순서와 문서간의 연관성을 알기 위해서)

예를 들어 XML문서 #1의 Key가 "100"이라면 XML문서 #2는 "100"이라는 값을 가지고 있어야 한다. 이런 방법을 사용함으로써 XML문서 #2가 XML문서 #1에 의해서 작성된 것이라는 사실을 알 수 있다.

※룰(rules) 표기

① XML문서 #1는 Processor A의 결과물이다.

② XML문서 #2는 Processor B의 결과물이다.

③ Processor B는 반드시 Process A다음에 수행을 해야 한다.(물론 Process A가 정상적으로 수행된 다음 XML문서 #1를 가지고 수행한다.)

XML문서가 가지고 있는 DTD는 아래의 내용과 같다.

<!ELEMENT Document(Process_ID | Parent_Process_ID | Child_Process_ID) >
<!ELEMENT Process_ID (#PCDATA) >
<!ELEMENT Parent_Process_ID (#PCDATA) >
<!ELEMENT Child_Process_ID (#PCDATA) >

①Process_ID : XML문서를 생성한 현재 프로세스의 식별자

②Parent_Process_ID : 현재의 프로세스를 수행하기 위해서 선행되어야 하는 프로세스의 식별자

③Child_Process_ID : 현재의 프로세스 다음에 수행할 자식 프로세스의 식별자

<Porlog>

rule 1 : XML문서#2 :- exit(XML문서#1)

rule 2 : Create(Processor A , XML문서#1)

rule 3 : Create(Processor B , XML문서#2)

즉, XML문서#2가 생성될 때 먼저 Prolog 엔진에게 질의한다. 이 때 Prolog는 XML문서#1이 작성되어 있는지를 검색하고 주어진 룰(rules)과 사실(fact)을 근거로 결과 값을 반환한다. 그 결과가 참이 되면 XML문서#2를 생성한다. 만약 그 결과가 거짓이라면 DTD에 정의된 내용을 확인하고 이전 프로세스(Processor A)의 정상적인 수행 여부를 검사한다. 수행 결과에 따라 복원(Rollback)과 수행(commint)을 진행한다.

## 3. 결론 및 추후 연구

비즈니스 룰을 기반으로 하는 시스템에서 룰을 분석하고 룰에 따른 프로세스의 진행은 필수적이다. 이런 비즈니스 룰 분석을 논리 프로그램인 Prolog를 통해서 분석하고 제어함으로써 보다 효율적인 시스템 설계가 가능 할 것이다. 차후에는 비즈니스 룰 자체를 논리 프로그램을 통해서 정의하고 수정할 수 있는 시스템의 설계와 구현이 필요할 것이다.

## 4.참고 문헌

[1]Amit Sheth, An Overview of WorkFlow Managerment From Process Modeling to WorkFlow Automation Infrastructure, 1995.

[2]Principles of Transaction Processing, Philip A. Bernstein

[3]Programming for artificial intelligence. Ivan Bratko