BPM 환경에서의 제약조건 관리

박철순[1] 최인준[2]
[1] 창원대학교 산업시스템공학과 / [2] 포항공과대학교 산업공학과

ABSTRACT
비즈니스 프로세스 관리(BPM)는 프로세스의 모델링, 분석, 실행에 이르는 비즈니스 프로세스 전 라이프 사이클을 지원하고자 하는 개념으로, 급변하는 경영 환경의 변화에서 기업의 경쟁력을 재고하기 위하여 필요로 되는 새로운 기업 컴퓨팅 패러다임이다. 비즈니스 프로세스 관리의 개념을 도입함으로써 기업들은 조직의 프로세스를 능동적이고 빠르게 (재)설계하고 수행할 수 있으며, 기업간 M&A 및 전략적 제휴를 효율적으로 수행할 수 있다. 기업간 협업이 이루어지기 위해서는 그들 간의 프로세스의 교류 및 통합이 필수적이다. 통합의 한 수단으로 여러 프로세스에 관련된 Data들을 Rule의 형태로 표현하여 관리할 수 있다.

본 연구에서는 워크플로, BPMS 등의 환경에서 비즈니스 Rule을 포함하는 비즈니스 프로세스 수행시에 관리해야 할 제약조건을 통합 관리할 수 있는 방법론을 제시한다. 비즈니스 프로세스를 수행하면서 체크해야 할 제약조건들을 규명한후 이들을 표현할 수 있는 프로세스 기반의 트리거 모델을 제시한다. 또한 본 방법론의 실현 가능성을 보여주기 위해 예제와 함께 프로토타입 시스템을 구현하였고 Performance 측면에서는 제한적이기는 하지만 DB에 기반한 제약조건 관리방법과의 비교를 통해서 본 연구에서 제안하고 있는 방법의 우수성을 보여준다. 본 연구에서 제안하는 방법론은 Workflow 및 BPMS와 같은 프로세스 관리 시스템에서 통합의 수단으로 사용되고 있는 EAI 등을 보완해 줄수 있는 툴로써 활용가능하리라 기대된다.

Keyword: Business Process Management, Workflow, Business Process Constraints, ECA Model

1. 연구배경

데이터베이스에 저장되어 있는 데이터의 건전성을 관리하기 위해서 데이터베이스 트리거를 사용하여 왔다. 하지만 이런 데이터베이스 트리거는 여러 개의 비즈니스 프로세스 또는 단위업무에 관련된, 즉 필수적으로 여러 데이터 아이템을 포함하고 있는 복잡한 비즈니스 프로세스 제약조건들을 관리하기에는 적절하지 못하다. 제약조건을 결정하는 각 데이터 아이템에 각각 데이터베이스 트리거를 정의할 수 있다. 이후 각 데이터 아이템의 값이 바뀔 때 매번 트리거 조건을 체크해서 재고정책 조건이 만족하지 않게되면 데이터 값을 바꿀수 없게 Reject 또는 Rollback과 같은 DB 트리거의 액션을 이용해서 관리할 수 있다.

하지만 위와 같은 DB 트리거를 이용하는 방법은 문제를 초래할 수 있다. 먼저, DB 트리거를 이용해서 제약조건을 관리하는 경우 관련된DB 테이블들이 분산된 이질적인 데이터베이스에 존재한다면 관리하기가 쉽지 않다. 둘째, 관련된 테이블의 값이 변할 때 마다 데이터베이스 트리거를 호출해야 하므로 심각한 부하를 초래할 수 있다. 더욱이 어떤 한 트리거의 액션은 다른 트리거를 호출하게 하고 이것은 시스템의 궁극적인 성능을 심각하게 떨어트릴수 있다. 복잡한 비즈니스 프로세스에 관련된 비즈니스 프로세스 제약조건을 관리하기 위해 DB 트리거를 사용하면 더욱 심각한 문제가 발생할 수 있다. 어느 기업에서 다음의 시나리오와 같이 재고수준을 관리하고 싶다고 가정하자. 이 기업에서는 생산지시를 내리거나 파손된 재고품이 발견되었을 때 재고수준을 체크해서 조치를 취하고 싶다고 가정하자. 생산지시를 내리는 시점에서 위의 조건이 위반되어 있음이 발견되면 재고수준을 만족시키기 위해서 더 많은 생산지시를 내린다. 파손된 제품이 발견되었을 때는 생산리드타임이 충분하지 않으면 긴급외부구매를 내는 액션을 취하고 싶다. DB 트리거의 경우 단위 트랜잭션이 수행될 때 마다 DB 트리거가 호출된다. 더욱이 DB 트리거를 통해서 행할 수 있는 액션은 트랜잭션을 취소하거나 메시지를 보내는 정도이다. 이러한 시나리오는 DB 트리거가 호출되는 시점과 비즈니스 프로세스 제약조건을 실제로 체크해서 조치를 취하는 시점 사이에는 시간갭이 발생함을 드러낸다. DB 트리거는 사람에 의해 수행되는 실질적인 비즈니스 프로세스 로직에 대해서는 충분히 유연하게 대처할 수 없다. DB 트리거의 더욱 심각한 단점은 제약조건이 위반되었을 때 수행해야 할 비즈니스 프로세스 또는 업무를 표현할 수 없다는 점이다. 따라서 비즈니스 프로세스 제약조건을 효율적으로 표현하고 관리할 수 있는 새로운 방법이 필요하다.

웍플로우 관리시스템 (WfMSs)은 이러한 비즈니스 프로세스에 관련된 복잡한 제약조건을 표현하고 관리하기 위한 하나의 좋은 대안이다. 하지만 이들은 후속 프로세스 선정 또는 후속업무 선정을 위한 분기조건, 시간 제약조건과 같은 프로세스 컨트롤에 관련된 제약조건에 초점을 맞추고 있다. 트랜잭션널 워크플로우는 비즈니스 프로세스의 수행에 관련된 좀더 일반적인 조건들을 다루고 있다 (Choi et al., 2002, Kamath and Ramamritham, 1998). 하지만 이들은 비즈니스 프로세스에 관련된 일반적인 규칙 또는 정책 같은 제약조건들 보다는 수행시 에러에 대한 처리에 초점을 맞추고 있다.

본 논문에서는 분산 이질적인 환경에서 복잡한 비즈니스 프로세스 제약조건을 표현할 수 있고 관리가 가능한 프로세스 기반의 트리거 (BPTrigger) 메커니즘을 제안한다. 먼저 포괄적인 지원을 위해 비즈니스 프로세스에 관련된 가능한 제약조건들의 카테고리를 찾아낸다. 그리고 이들을

수행시점에 따라서 entry, exit, running 및 invariant 와 같이 네가지로 분류한다. 이렇게 분류된 제약조건을 관리하기 위해 비즈니스 프로세스 관점에서 ECA 모델을 확장하여 BPTrigger를 정의하고 이것의 Syntax를 제공한다.

논문은 다음과 같이 구성되어 있다. 먼저 2절에서는 제약조건 관리에 관련된 연구들에 대해서 살표본다. 3절에서는 비즈니스 프로세스 제약조건을 분류를 제안하고 이를 표현하고 관리하기 위한 BPTrigger를 제안한다. 4절에서는 벤치마킹과 실현가능성을 보이기 위해 개발한 BPTrigger 프로트타입을 소개한다. 마지막으로 추후연구과제와 함께 결론을 맺는다.

2. Related Work

다양한 비즈니스 응용프로그램과 관련된 객체들이 기업의 비즈니스 프로세스에 관계하기 때문에 이들 사이의 제약조건들은 기존의 존재하던 문제들을 더욱 복잡하게 만든다. 비즈니스 프로세스모델링에서 고려하는 비즈니스 프로세스 제약조건들은 비즈니스의 구조에 대한 구조적인 것들과 비즈니스에서 수행해야할 조치들 , 건전성에 대한 규칙들, 통합요구사항등을 포함한다. Weiden(2000)은 비즈니스 규칙에 대해 포괄적인 분류를 제안했다. 하지만 이러한 작업들은 주로 업무조정 및 업무흐름을 위한 조건체크에 초점을 맞추고 있다. 비즈니스 프로세스 제약조건은 업무수행자에 의한 액션이 업무매뉴얼 또는 기업의 정책에 부합해야 한다는 점을 확인해 줄수 있어야 한고 업무수행자가 업무를 수행하는 것을 지원해 주어야 한다.

워크플로우 관리는 비즈니스 프로세스 관리를 위한 주요한 정보시스템의 하나로써 인식이 되고 있다. 특히 트랜잭셔널 웍플로우 분야에 많은 연구들이 이루어 지고 있다 (Choi et al., 2002). 비즈니스 프로세스 관리를 위한 좀더 포괄적인 기술이 비즈니스 프로세스 관리 (BPM) 이다. BPM에 대한 표준을 위해 조직된 BPMI는 인터넷상에서 비즈니스 당사자들, 기업의 부서들, 다양한 응용프로그램들을 어우르는 비즈니스 프로세스의 관리를 위한 공개규약(open specifications)을 개발하는 것에 초점을 맞추고 있다.

3. BPTrigger for Enforcement of Business Process Constraints

이장에서는 비즈니스 프로세스 제약조건의 포괄적인 분류를 제공한다. 이 분류를 바탕으로 제약조건을 표현하고 실행하기 위한 BPTrigger의 문법을 제시한다.

3.1. Business Process Constraints

비즈니스 규칙을 규명해 내어 분류하기위해 많은 연구들이 수행되어 왔다. 특히 Weiden(2000)은 비즈니스 규칙을 Structural, Behavioral, Managerial과 같이 세 가지의 카테고리로 분류되어야 한다고 주장하였다. 본 연구에서는 문헌조사 및 연구경험을 토대로 정보시스템의 관점에서 Object, Behavior, Execution, Integration, Translation 및 Managerial 과 같이 여섯 개의 카테고리로 구분을 하였다. 이러한 제약조건 카테고리는 업무를 수행시 제약조건을 체크하는 시점에 따라 다시 Entry, Exit, Running 및 Invariant 와 같이 재구분을 하였다.

3.2. BPTrigger: A Process-oriented Trigger

앞 절에서 제시한 카테고리의 제약조건들을 관리하기 위해서 비즈니스 프로세스 관점에서 ECA모델을 확장하여 BPTrigger 모델을 제안하였다. ECA에 기반을 둔 다른 시스템과는 달리 BPTrigger는 비즈니스 프로세스가 수행될 때 제약조건을 관리하기 하기 위해 비즈니스 프로세스 관리 (BPM) 개념을 이용하고 있다. BPTrigger는 사용자가 데이터 아이템이 아닌 관련된 비즈니스 프로세스 또는 단위업무에 제약조건을 정의하여 프로세스 수준에서 제약조건 위반 또는 예외상황에 대처할 수 있게 한다. 아래 그림1에 BNF 표기를 이용해서 BPTrigger의 최상위 구조를 나타내고 있다.

BPTrigger의 다음과 같은 컴포넌트로 구성되어 있다:

1) Administrative information: <trigger-name>
이 컴포넌트는 각 BPTrigger를 구분하기 위해 사용된다.

2) Triggering event: <activation-time> <event-description>
이 컴포넌트는 관련된 BPTrigger를 호출할 이벤트와 실행할 시점을 표시한다. DB 트리거와는 달리 <event-description> 에 표시되는 이벤트는 프로세스, 단위업무, 또는 특정한 프로세스/단위업무 인스턴스를 포함할 수 있다. <process-event> 가 특정한 인스턴스 일때는 정의된 BPTrigger는 그 인스턴스에만 적용된다. 프로세스나 단위업무에 대해 정의될때는 정의된 BPTrigger는 해당 프로세스 또는 단위업무의 모든 인스턴스에 적용된다. <activation-time>에는 제약조건을 실행할 시간을 지정한다. 여기에 지정될 수 있는 키워드는 다음과 같다:

- BEFORE 는 관련된 프로세스 또는 단위업무가 수행되기 전에 제약조건을 체크하는 것을 의미한다.
- AFTER 는 관련된 프로세스 또는 단위업무가 수행된 직후에 제약조건을 체크하는 것을 의미한다.
- WHEN 은 업무가 수행되는 임의의 시점에 제약조건을 체크하는것을 의미한다.

3) Trigger condition: CONDITION <condition-description>
이 컴포넌트는 정의된 BPTrigger에 대해 평가해야할 조건식을 포함한다.

4) Triggered action: ACTION <action-description>
이 컴포넌트는 조건식이 참일 때 수행해야할 트리거의 액션을 포함한다. BPTrigger에서 액션은 수행한 프로세스 또는 단위업무를 다시 수행하거나 수행결과를 보정하는 다른 프로세스 또는 업무가 될 수 있다. 이러한 액션들은 예를들면 비즈니스 프로세스 관리 시스템(BPMS)의 수행 리스트에 삽입할 수 있다.

5) Triggered exception handler: EXCEPTION <exception-description>
이 컴포넌트는 예외상황이 발생할시 수행할 액션을 지정한다.

6) Trigger priority: PRIORITY <number>
이 컴포넌트는 정의된 BPTrigger의 우선순위를 지정할 수 있다. 여러 개의 BPTrigger들이 동일한 우선순위를 가질 때 우선순위는 이벤트의 시스템 도착시간에 따라 결정된다.

```
<BPTrigger-def> = DEFINE BPTrigger <trigger-name> AS
                  [ BEFORE | AFTER| WHEN ] <event-description>
                  CONDITION <condition-description>
                  ACTION <action-description>
                  EXCEPTION <exception-description>
                  PRIORITY <number>;
<event-description> = <process-event> | <temporal-event> | <user-event>;
<process-event> = <Process> | <Activity> | <Instance>;
<condition-description> = <relational-expression> | <set-expression> | <and-rel> | <or-rel>;
<action-description> = <Process> | <Activity> [, <notify>];
<exception-description> = <Process> | <Activity> [, <notify>];
<Process> = letter, {letter | decimal digit };
<Instance> = letter, {letter | decimal digit };
<Activity> = letter, {letter | decimal digit };
```

<그림1> BPTrigger의 최상위 구조

BPTrigger의 운영 시나리오는 다음과 같다. 먼저 비즈니스 프로세스나 단위업무에 대해 BPTrigger를 정의한다. 같은 프로세스나 단위업무에 대해 여러 개의 BPTrigger가 정의될 수도 있다. 이벤트가 발생했을때 이벤트에 연결된 모든 BPTrigger들이 활성화 된다. 하나 이상의 BPTrigger가 동시에 활성화될때는 실행순서는 사용자가 부여한 우선순위에 따라 결정된다. 활성화 시점 (BEFORE, AFTER 또는 WHEN) 에서 트리거의 조건이 평가된다. 조건식이 참으로 판명되면 트리거 액션이 수행된다. 즉, 수행할 업무를 적절한 업무수행자의 Worklist에 삽입할 수 있다. 예외상황이 발생할때는 예외상황 컴포넌트에 정의된 액션을 수행하게 한다.

4. BPTrigger 프로토타입 시스템

이 장에서는 BPTrigger의 프로토타입 시스템에 대해 소개한다.

4.1. 예제

1장에서 소개된 재고정책예제를 이용해서 BPTrigger의 실현가능성을 보여준다. 그림 2는 재고수준에 대한 BPTrigger를 나타내고 있다. 제약조건이 위반되고 동시에 파손된 부품이 존재할 때 취할 액션을 위해 그림 2-b와 같은 추가적인 BPTrigger가 필요하다.

```
DEFINE BPTrigger Inventory_Level01 AS
BEFORE production order generation
CONDITION ((current stock + receipts + purchase_order) LT
                        total_demand_qty)
ACTION emergency production order
            Notify = cspark@postech.ac.kr
EXCEPTION check inventory level
            Notify=mkjang@postech.ac.kr
PRIORITY 2;
```

a) Before performing production order generation activity

```
DEFINE BPTrigger Inventory_Level02 AS
AFTER inventory counting
CONDITION ((current stock + receipts + purchase_order) LT
                        total_demand_qty)
ACTION emergency production order
            Notify= cspark@postech.ac.kr
EXCEPTION check inventory level
            Notify=mkjang@postech.ac.kr
PRIORITY 2;
```

b) After performing inventory counting with result of no damaged stock

<그림 2> 재고수준을 위한 BPTrigger

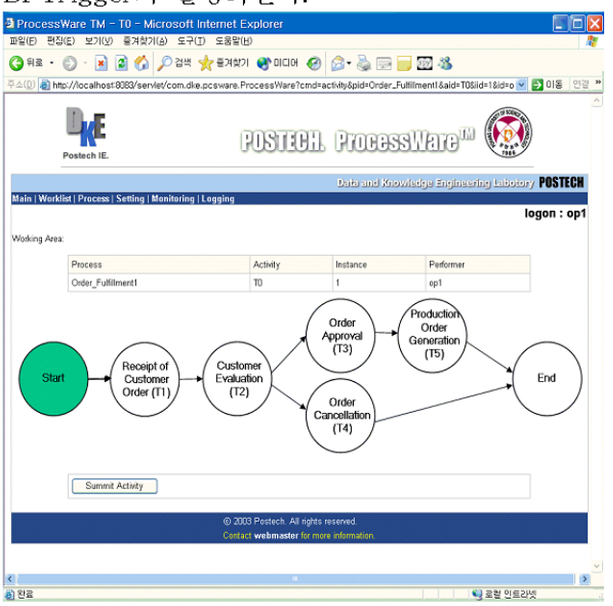
4.2. XML DTD for BPTrigger

프로토타입 시스템에서 정의된 BPTrigger는 XML로 변환되어 사용된다. XML을 사용함으로써 XML의 타당성 검사 및 호환성에 대한 장점을 활용하여 쉽게 구현할 수 있다. 이것은 또한 다양한 플랫폼에서 다양한 형태로 비즈니스 프로세스 제약조건을 정의하고 교환할 수 있게 한다. XML로 표현된 BPTrigger는 조직사이에서 쉽게 교환되어 사용될 수 있고 사용자의 요구에 따라 다양한 형태로 쉽게 변활될 수 있다.

4.3. 프로토타입 시스템

전형적인 DB 트리거와의 성능비교를 하고 제안된 방안의 실현가능성을 보여주기 위해서 프로토타입 시스템을 개발하였다. 프로토타입 시스템은 MicroSoft Visual C++ 6.0과 MicroSoft SQL Server를 이용해서 개발되었다.

본 논문에서는 BPTrigger 시스템을 프로토타입 BPMS인 ProcessWare상에서 서브모듈로 운용된다. 그림 3은 수행할 프로세스를 나타내는 화면을 나태난다. 주문이 도착했을 때 표준생산지시가 내려진다. 업무를 수행하기 전에 생산지시생성업무의 사전조건으로 정의된 재고수준에 대한 BPTrigger가 활성화된다.



<그림 3> Snapshot of BPTrigger 수행

4.4. 성능 벤치마킹

벤치마킹은 동일한 예제 프로세스에서 같은 제약조건에 대해 DB 트리거를 사용한 경우와 BPTrigger를 사용한 경우의 걸린 시간을 비교함으로서 수행하였다. 가상의 주문이 들어오면 이를 처리하기 위해 프로세스가 만들어 지고 주문이 생성된다. 테스트는 Oracle9i 와 MicroSoftSQL Server 2000 데이터베이스에서 각각 수행하였다.

[표1] 벤치마킹 결과

Result (Sec) / Data	DB Trigger			BPTrigger		
	No. of Triggers			No. of Triggers		
	10	100	1000	10	100	1000
1000	0.048/0.029	0.27/0.095	2.2/1.37	0.051/0.032	0.051/0.032	0.052/0.033
10000	0.181/0.054	1.0/0.097	9.0/1.38	0.052/0.032	0.052/0.033	0.052/0.035
100000	6.5/0.8	95/1.2	1185/3.7	0.1/0.08	0.3/0.1	0.3/0.2

수행결과의 일부분은 표 1에 요약되어 있다. 기대했던 것처럼 DB 트리거의 경우 트리거 개수와 데이터의 양이 증가함에 따라 DB 트리거 에 수행에 걸린 시간이 증가함을 보이고 있다. 반면 BPTrigger의 경우 거의 영향을 받지 않고 있음을 보이고 있다. 따라서 BPTrigger를 사용함으로써 전형적인 DB 트리거에 비해 훨씬 적은 부담을 가지고 제약조건을 체크할 수 있음을 알 수 있다.

5. 결론 및 추후연구과제

본 연구에서는 기업에서 중요한 지적자산으로 관리할 필요가 있는 비즈니스 프로세스를 표현하고 체크하기 위한 메커니즘으로 BPTrigger를 제안하고 있다. 먼저 문헌조사를 기반으로 제약조건을 규명해 낸 후 체크시점에 따라 entry, exit, running 및 invariant 와 같이 분류하였다. 이러한 제약조건을 표현하고 관리하기 위해서 비즈니스 프로세스 관점에서 ECA모델을 확장하여 BPTrigger를 정의하였고 이의 문법을 제시하였다. 성능평가와 실현가능성을 보여주기 위해서 BPTrigger 프로토타입을 구현하였다.

본 연구는 분산환경에서의 룰기반의 프로세스 지향의 통합에 활용될 수 있다. EAI와 같이 분산된 응용프로그램간의 협업 또는 통합을 지원하기 위해 지원툴로서 활용할 수 있다. 추후연구로는 웹서비스와 같은 최신기술을 이용해서 BPTrigger 관리시스템을 구현할 필요가 있다. 추가 연구과제로는 제약조건들간의 충돌을 회피하거나 방지하는 연구가 필요하다. 마지막으로 트리거 기반의 시스템에서 제시할 필요가 있는 Termination에 대한 연구가 필요하다.

Acknowledgement

The authors would like to thank the Ministry of Education of Korea for its financial support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program.

References

Choi, Injun, Chulsoon Park, and Changwoo Lee, 2002, Task-Net: Transactional Workflow Model based on Colored Petri Net, European Journal of Operational Research, Vol. 136, 383-402

Choi, Injun, Minseok Song, Chulsoon Park and Namkyu Park, 2003a, An XML-based process definition language for integrated process management, Computers in Industry, 50, 85-102

Kamath, Mohan and Krithi Ramamritham, 1996, Correctness Issues in Workflow Management, Distributed Systems Engineering, Vol. 3, 213-221

Weiden, M., 2000, A Critique of the Pure Business-rule Approach, Department of Social Science Informatics, University of Amsterdam