객체지향 데이터베이스를 이용한 전문가 시스템 지식의 지속성 유지

민 미 경[1)]

I 요약

전문가 시스템의 응용분야가 크고 복잡해짐에 따라, 대량의 지식을 공유하고 저장하기 위하여 데이터베이스를 이용하는 기법이 활발하게 연구되고 있다. 본 논문에서는 전문가 시스템 지식을 객체지향 데이터베이스에 지속적으로 저장 관리하는 방법을 제안한다. 전문가 시스템은 객체지향 데이터베이스 관리시스템과 밀접하게 결합되며, 전문가 시스템의 지식은 모두 객체형태로 데이터베이스에 저장되어 데이터베이스의 객체들과 동일하게 관리된다.

II 서론

최근 전문가 시스템에 적용되는 응용분야의 크기와 범위가 증가함에 따라 전문가 시스템의 지식베이스가 점차 대형화하고 복잡해짐에 따라 기존의 전문가 시스템만으로는 개발이 어려운 응용분야들이 발생하기 시작했다. 이러한 분야에서 요구되는 지식을 관리하기 위해서는 지식이 지속적으로 저장될 수 있어야 하며 다수의 응용 프로그램이나 사용자에 의해서 공유될 수 있어야 한다.. 즉, 데이터베이스 시스템의 특성인 보안, 동시성 제어, 저장, 추울 등의 기능에 대한 필요성이 지식기반 시스템에서도 요구된다. 한편 데이터베이스 시스템에서도 지식기반 시스템의 추론이나 결정과 같은 지능적인 기능을 필요로 하게 되었다. 따라서 두 시스템의 결합에 관한 연구가 활발히 계속되어 왔다.

데이터베이스 시스템을 중심으로 전문가시스템을 이용하는 예는 추기 논리 프로그래밍에 기초한 시스템의 연구에서 시작하며, 최근에는 일반적인 형태의 규칙인 생성규칙(production rule)을 이용한 시스템들의 연구와 개발에 중점을 두고 있다. 이러한 시스템들은 관계형 또는 객체지향 데이터베이스 시스템을 기반으로 하며, 활동적 데이터베이스의 트리거로서 생성규칙을 사용한다. 즉, 데이터베이스에 특정한 변화가 일어날 경우 수행하여야 할 동작이 규칙으로 명시된다. 전문가시스템을 중심으로 데이터베이스 시스템을 이용하는 예는 느슨한 결합 시스템이나 밀접한 결합 시스템의 예가 있다. 그러나 이들은 지식베이스의 사실만을 데이터베이스에 저장하여 이용하고 지식베이스의 규칙은 지속적으로 저장하지 못한다. 규칙을 데이터베이스에 지속적으로 저장하기 위한 기법은 활동적 데이터베이스에서 연구되어 왔으나, 모두 데이터베이스의 관점에서 필요한 형태의 규칙저장과 운용에 관한 연

1 서울시 성북구 정릉동 산 16-1, 서경대학교 컴퓨터과학과
mkmin@bukak.seokyeong.ac.kr

구들이다.

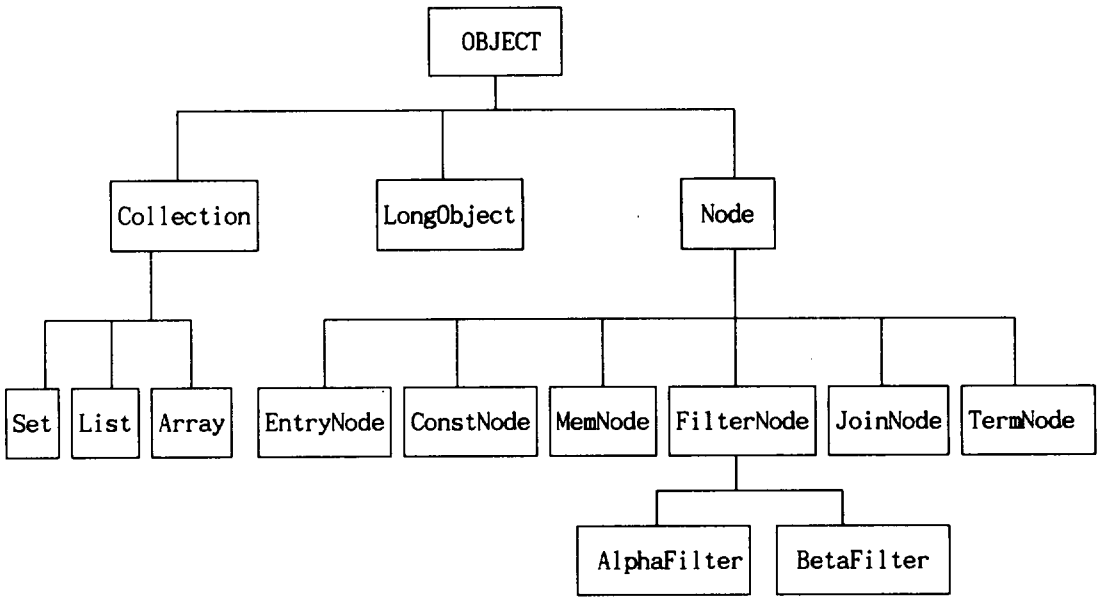
따라서 전문가 시스템에서 지식 전체를 데이터베이스에 저장, 운용할 수 있는 연구가 필요하다. 특히, 생성규칙은 추론과정에서 if-then 형태로 그대로 이용되기보다는 추론에 적합한 내부형태인 규칙망으로 컴파일되어 이용되므로, 데이터베이스와 연결 시 규칙을 그대로 데이터베이스에 저장하는 것보다는 컴파일된 규칙망 형태로 저장하는 것이 보다 효율적일 것이다. 또한 데이터베이스에 저장하고자 하는 규칙망은 노드와 링크로 이루어져 그 형태가 복합적이므로, 평면적인 데이터를 다루는 관계형 데이터베이스를 이용하는 것보다는 서로 복합적으로 연관된 지식을 다룰 수 있는 객체지향 데이터베이스를 이용하여 저장, 관리하는 것이 더 적합하다.

본 논문에서는 전문가 시스템을 객체지향 데이터베이스와 밀접하게 결합시켜 전문가 시스템 지식을 지속적으로 데이터베이스에 저장 관리할 수 있는 기법을 제시한다. 이 시스템은 전문가 시스템에서 지식베이스의 사실과 규칙을 모두 데이터베이스에 저장, 운용할 수 있도록 한다. 또한 추론기관이 데이터베이스의 클래스 라이브러리에 포함되므로 데이터베이스와 완전히 통합되어 있다. 다음 3장에서는 본 논문에서 제시하는 전문가 시스템 지식의 구조를 살펴보고 4장에서는 지식의 지속성을 유지하는 기법을 소개한다. 마지막으로 5장에서 결론을 맺는다.

III. 지식 구조

3.1 지식 계층구조

전문가 시스템의 지식베이스를 구성하는 지식은 크게 사실(fact)과 규칙(rule)으로 구분된다. 사실은 하나의 객체로서 클래스(class)와 인스턴스(instance)로 구분한다. 클래스는 하나의 개념을 나타내며 인스턴스는 실세계에 존재하는 개체와 대응된다. 클래스는 다시 시스템 정의 클래스와 사용자 정의 클래스로 구분된다. 시스템 정의 클래스에서는 시스템이 제공하는 연산을 정의하고 있으며, 이러한 연산은 하위 클래스로 차례로 계승되어 사용자 정의 클래스에서 사용된다. <그림 1>과 같이 클래스 계층구조의 시작은 OBJECT라고 하는 시스템 정의 클래스이다. OBJECT는 전체 객체들의 공통된 성질을 갖는 객체이며 모든 클래스는 OBJECT의 하위 클래스이다.



< 그 림 1> 지식 계층구조

3.2. 규칙망

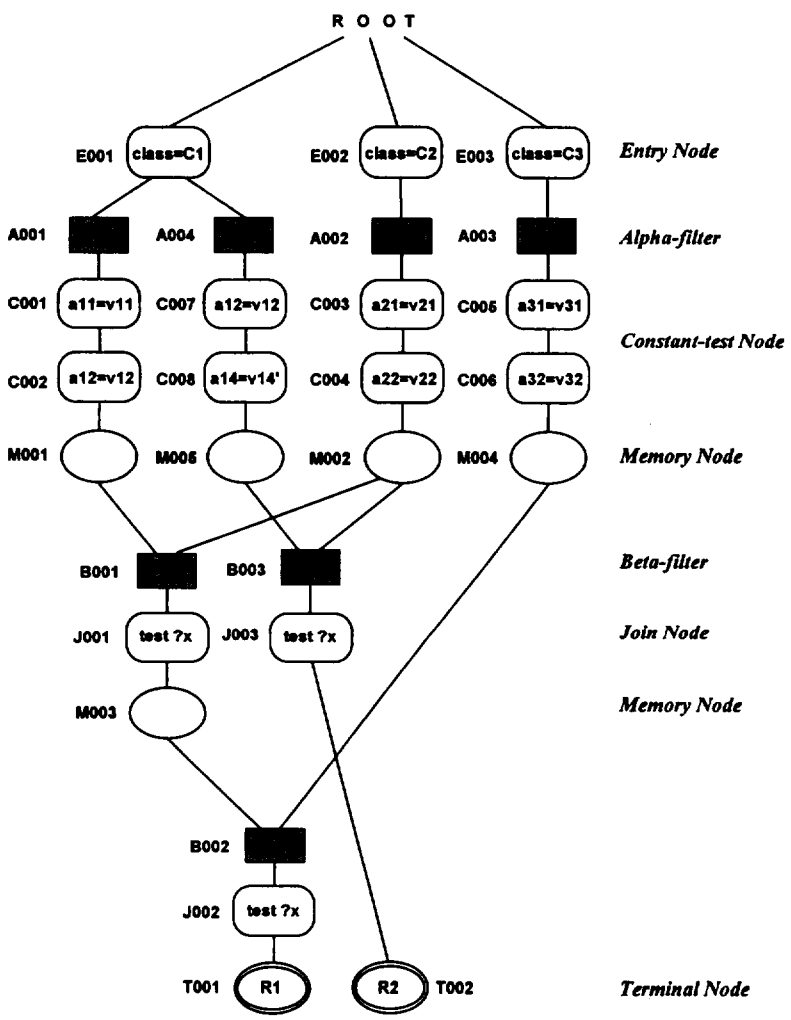
전문가 시스템 지식베이스의 규칙들은 컴파일되어 규칙망(rule network)을 형성한다. 이 규칙망은 객체지향 방식으로 형성, 실행된다. 규칙망은 노드(node)와 노드를 연결하는 링크(link)로 구성되며, 각 노드는 하나의 객체를 의미한다. 다음은 C1, C2, C3 3개의 사용자 정의 클래스와 O1, O2, O3 3개의 인스턴스, R1, R2, 2개의 규칙으로 이루어진 지식베이스의 예이다. 각 클래스 내에는 애트리뷰트(attribute)들이 선언되어 있다. R1과 R2는 패턴으로 이루어진 if 부분과 객체의 애트리뷰트 값을 변화시키는 then 부분으로 구성되어 있다. 이러한 지식베이스의 규칙을 컴파일하면 <그림 2>와 같은 rule network이 생성된다.

< 예 1 >

```
Classes
  CLASS C1: virtual public OBJECT
  { a11; a12; a13; a14; a15; };
  CLASS C2: virtual public OBJECT
  { a21; a22; a23; };
  CLASS C3: virtual public OBJECT
  { a31; a32; a33; };
Instances
  O1 {v11 v12 v13 v14 v15}    /* an instance of C1 */
  O2 {v21 v22 v13}            /* an instance of C2 */
  O3 {v31 v32 v13}            /* an instance of C3 */
```

Rules

```
RULE R1   {
      for : (?o1 in C1)
            (?o2 in C2)
            (?o3 in C3)
       if : (?o1 a11 = v11  a12 = v12  a13 = ?x)
            (?o2 a21 = v21  a22 = v22  a23 = ?x)
            (?o3 a31 = v31  a32 = v32  a33 = ?x)
    then : (modify ?o1 a14 = v14')
}
RULE R2   {
      for : (?o1 in C1)
            (?o2 in C2)
       if : (?o1 a12 = v12  a14 = v14'  a13 = ?x)
            (?o2 a21 = v21  a22 = v22   a23 = ?x)
    then : (modify ?o1 a15 = v15')
}
```



< 그 림 2 > 규칙망의 예

노드는 크게 입구노드, 상수검사노드, 메모리노드, 결합노드, 알파필터노드, 베타필터노드, 말단노드 등의 6가지 종류로 구분된다. 입구노드는 규칙망의 최상층 노드로서 클래스의 이름을 검사한다. 상수검사 노드는 애트리뷰트의 상수값을 검사하며 메모리 노드는 규칙의 패턴의 부분매칭 정보를 저장한다. 결합노드는 패턴 간의 일관성을 검사한다. 필터노드는 하위노드에서 매칭이 더 이상 필요한지를 판단하는 노드이다. 필터노드에는 알파필터 노드와 베타필터 노드가 있다. 알파필터 노드는 상수검사 노드의 매칭이 필요한가 검사하며 베타필터 노드는 결합노드에서 매칭이 필요한가 검사한다. 말단노드는 규칙을 만족키는 객체의 집합 및 기타 규칙에 관한 정보를 포함하고 있다.

각 종류의 노드에 대한 공통된 구조 및 동작은 EntryNode, ConstNode, MemNode, JoinNode, FilterNode, AlphaFilter, BetaFilter, TermNode라고 하는 시스템 정의 클래스에 정의된다. 이들은 <그림 1>에서와 같이 Node라는 상위 클래스로부터 모든 노드에 대한 공통된 성질을 계승받으며, <그림 3>과 같은 내부구조를 갖는다.

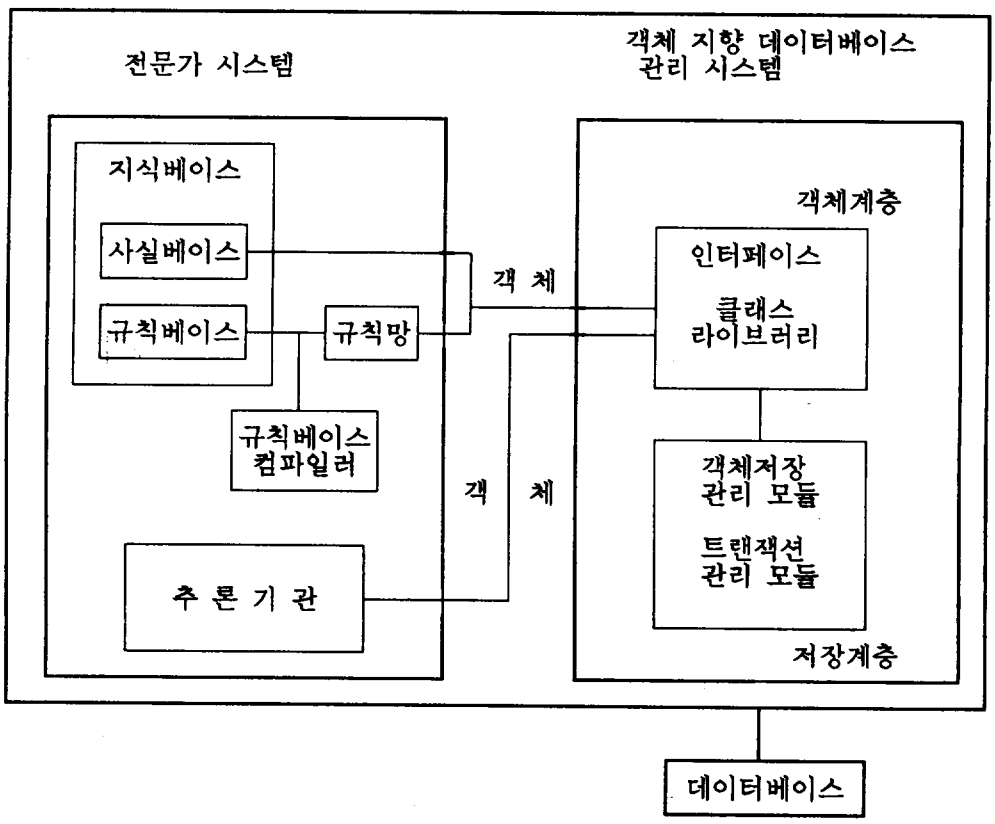
```
CLASS Node : <superclass> OBJECT {
  <attributes>
    outLink;
  <methods>
    attach(Node *link);
    virtual equalNode(Node *node1, Node *node2);
    virtual evaluate(Token *x);
    virtual evaluate(TokenProduct *X);
}
CLASS EntryNode : <superclass> Node {
  <attributes>
    className;
  <methods>
    generate(string className);
    evaluate(Token *x);
    equalNode(Node *node1, Node *node2);
}
CLASS ConstNode : <superclass> Node {
  <attributes>
    attrName;
    operator;
    valueToBeTested;
  <methods>
    generate(string attrName, int operator, Value valueToBeTested);
    evaluate(Token *x);
    equalNode(Node *node1, Node *node2);
}
CLASS AlphaFilter: <superclass> FilterNode {
  <methods>
    evaluate(Token *x);
}
CLASS BetaFilter : <superclass> FilterNode /* 생략 */
CLASS MemNode : <superclass> Node /* 생략 */
CLASS JoinNode : <superclass> Node  /* 생략 */
CLASS TermNode : <superclass> Node /* 생략 */
```

< 그 림 3 > 규칙망의 노드구조

Ⅳ. 지식의 지속성 유지 기법

4.1. 전문가 시스템과 객체지향 데이터베이스의 연결

객체지향 데이터베이스와 연결되는 전문가 시스템의 전체 구조는 <그림 4>와 같다. 전문가 시스템은 개발 도중 지식 공학자들이 입력한 지식베이스를 이용하여 추론을 행하며, 전향 추론 방법에 의해 매칭-선택-실행 과정을 반복한다. 추론기관은 추론에 의해 지식을 생성, 수정, 삭제할 수 있으며, 이는 다시 지식베이스에 반영된다. 지식베이스는 서로 협력하에 있는 다수의 지식 공학자에 의해 공유되며, 지식 공학자들은 추론을 하면서 지식을 구성하여 응용 시스템을 개발한다. 지식베이스 내의 사실은 객체 지향 데이터베이스의 객체와 동일하다. 규칙베이스는 규칙 컴파일러에 의해 객체들의 집합인 규칙망으로 변환된다. 객체지향 데이터베이스와의 연결은 객체 계층에서 이루어진다. 객체들의 집합인 규칙망과 사실들은 객체 계층으로 보내지면서, 데이터베이스의 객체와 동일하게 관리된다. 즉, 이후로는 생성, 수정, 삭제된 모든 지식들이 객체 형태로 데이터베이스에 저장되며, 추론기관은 이 객체를 이용하여 추론을 한다. 전문가 시스템 내의 지식베이스는 지식 공학자에게 보여지는 형태로서만 존재한다.



<그 림 4> 전문가 시스템과 객체지향 데이터베이스의 연결구조

4.2 지속적 지식의 생성

지속적 지식의 생성을 보이기 위하여 <예 1>의 클래스 C1이 클래스로 정의되어 클래스 라이브러리에 저장되는 예를 보면 다음과 같다. 사용자가 정의한 클래스는 OBJECTS 클래스의 하위 클래스가 된다.

```
CLASS C1: virtual public OBJECTS {
    string a11;
    string a12;
    string a13;
    string a14;
    string a15;
}
```

클래스 C1에 속하는 인스턴스 O1은 사용자에 의해 다음과 같이 정의된다.

```
Database *factDB = Database::open("Sample");
Transaction::begin();
schema C1 *O1;
O1 = new <factDB> C1("v11", "v12", "v13", "v14", "v15");
O1->makePersistent();
Transaction::commit();
factDB->close();
```

인스턴스는 데이터베이스에 저장되는 객체이므로 이를 정의하기 위해서는 먼저 트랜잭션을 시작해야 하며, 정의를 마친 후에는 트랜잭션을 완료해야 한다. 예를 들어 C1 클래스에 속하는 O1이라는 하나의 인스턴스를 정의하기 위해서는 먼저 C1 타입의 변수를 하나 선언한 후, 애트리뷰트의 값들을 지정하여 객체를 주기억장치에 생성한다. 이 때 OID가 지정된다. makePersistent라는 메소드의 호출은 주기억장치에 정의된 임시 객체를 트랜잭션이 완료될 때 지속적 객체로 만든다는 것을 의미한다.

규칙컴파일러는 지식베이스의 규칙을 컴파일하여 규칙망의 노드를 생성한다. 다음은 <예 1>의 규칙 중 R1을 컴파일 하여 생성하는 지속적 규칙망의 일부이다.

```
Database *nodeDB = Database::open("Sample");
Transaction::begin();
schema EntryNode *en1;
en1 = new <nodeDB> EntryNode();
en1->generate("C1");
schema AlphaFilter *an1;
an1 = new <nodeDB> AlphaFilter();
an1->generate(a11, a12, a13);
schema ConstNode *cn1;
cn1 = new <nodeDB> ConstNode();
cn1->generate(a11, =, "v11");
/* ... 나머지 노드 생성을 위한 코드 */
```

```
Transaction::commit();
nodeDB->close();
```

4.3 알고리즘

객체는 임시 객체(temporary object)와 지속적 객체(persistent object)로 구분할 수 있다. 지속적인 객체는 트랜잭션이 성공적으로 끝난 후 데이터이타베이스에 반영되어, 다른 트랜잭션이나 응용 프로그램에서 사용할 수 있다. 지속성은 여러 응용 프로그램이나 사용자들에게 객체의 공유성(sharability)을 제공한다. 임시 객체는 하나의 트랜잭션 내에서만 사용가능하며, 트랜잭션이 끝남과 동시에 소멸되어 데이터베이스에 반영되지 않는다. 객체는 항상 임시 객체로 생성되며, OBJECT 클래스의 makePersistent 메소드에 의해 지속적 객체로 변환된다. 이 때, 이 지속적 객체가 참조하는 모든 객체도 지속적 객체로 변환된다. 지속적 상태로 바뀐 객체는 트랜잭션이 성공적으로 끝나면 데이터베이스에 저장된다.

데이터베이스의 객체를 생성, 추출, 수정, 삭제하기 위해서는 데이터베이스를 open한 후, 작업이 끝나면 close한다. 작업은 여러 개의 트랜잭션으로 이루어질 수 있다. 같은 프로그램에 의해 시스템은 먼저 Person 클래스에 속하는 p1이라는 임시 객체와 p2라는 임시 객체를 객체 계층의 객체 버퍼에 생성한다. 이 때, 각각에는 OID가 지정된다. 객체 계층에서 객체를 생성하는 알고리즘은 다음과 같다.

```
Algorithm CreateAtObjectLayer(O)
    if (there is no room in the ObjectBuffer) then
        SWO = select an object from the ObjectBuffer
                to be swapped out;
        swap out SWO to the swap-area;
    endif;
    create O in the ObjectBuffer;
end CreateAtObjectLayer;
```

프로그램의 마지막 Transaction::commit 문장에 의해 트랜잭션이 성공적으로 완료되면 객체 계층은 저장 계층으로 생성된 객체를 보내 지속적 객체로 만든다. 저장 계층에서의 객체 생성 알고리즘은 다음과 같다.

```
Algorithm CreateAtStorageLayer(O)
{
    create O in the disk;
}
```

다음은 각각 객체 계층과 저장 계층에서 객체를 추출하는 알고리즘이다.

```
Algorithm ReadAtObjectLayer(O)
    I = OID of O;
    if (O exists in the ObjectBuffer) then
        read O from the ObjectBuffer;
    else
        if (there is no room in the ObjectBuffer) then
```

```
            SWO = select an object from the ObjectBuffer
                  to be swapped out;
            swap out SWO to the swap-area;
        endif;
        O = ReadAtStorageLayer(I, ReadLock);
        copy O in the ObjectBuffer
    endif;
    return O;
end ReadAtObjectLayer;

Algorithm ReadAtStorageLayer(OID, lockType)
    P = page which contains O with OID;
    loop
        if P can get a lock of lockType
            read P from the disk;
    until (get a lock);
    return O;
end ReadAtStorageLayer;

Algorithm ModifyAtObjectLayer(O)
    OID = oid of O;
    if (O exists in the ObjectBuffer) then
        if (O is locked with WriteLock) then
            modify O in the ObjectBuffer;
        else
            O = ReadAtStorageLayer(OID, WriteLock);
            modify O in the ObjectBuffer;
        endif;
    else
        if (there is no room in the ObjectBuffer) then
            SWO = objects to be swapped out from the ObjectBuffer;
            swap out SWO to the swap-area;
        endif;
        O = ReadAtStorageLayer(OID, WriteLock);
        copy O to the ObjectBuffer;
        modify O in the ObjectBuffer;
    endif;
end ModifyAtObjectLayer;
```

객체 생성 시와 마찬가지로 프로그램의 마지막 Transaction::commit 문장에 의해 트랜잭션이 성공적으로 완료되면 객체 계층은 저장 계층으로 수정된 객체를 보내 지속적 객체로 만든다. 저장 계층에서의 객체 수정 알고리즘은 다음과 같다.

```
Algorithm ModifyAtStorageLayer(O)
    P = page which contains O with OID;
    modify O in P;
end ModifyAtStorageLayer;
```

Ⅳ. 결론

본 논문에서는 점차 크고 복잡해지는 응용분야에 적합한 전문가 시스템을 위하여 객체지향 데이터베이스를 이용하여 지식의 지속성을 유지하는 기법을 제시하였다. 제안된 시스템은 일반적으로 데이터베이스와의 통합 시 발생하는 결합 불일치 문제를 객체지향 방식이라는 개념으로 해결하였다. 지식은 OODB의 데이터와 동일한 형태와 의미를 가지므로 지식베이스와 데이터베이스는 자연스럽게 하나로 통합된다. 또한 사실뿐만 아니라 규칙도 데이터베이스에 저장되므로 지식 전체가 지속적이고 공유 가능하다는 장점이 있다.

참고문헌

[1] 김 일 곤, 박 창 현, 장 혜 진, 김 태 권, 민 미 경, *HEXPERT+* 기술보고서, 서울대학교 인공지능 연구실, 1991.

[2] 홍 은 지, 이 영 훈, 박 현 주, 이 주 홍, 전 용 석, *Obase* 기술보고서, 서울대학교 인공지능 연구실, 1992.

[3] Ballou, N., Chou, H., Garza, J., Kim, W., Petrie, C., Russinoff, D., Steiner, D., and Woelk, D., "Coupling an Expert System Shell with an Object-Oriented Database System," *Journal of Object-Oriented Programming*, 1988, pp. 12 - 21.

[4] Brownston, L., Farrell, R., Kant, E., and Martin, N., *Programming Expert Systems in OPS5*, Addison-Wesley, 1985.

[5] Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Journal of Artificial Intelligence*, Vol. 19, No. 1, pp. 17 - 37, 1982.

[6] Freundlich, Y., "Knowledge Bases and Databases: Converging Technologies, Diverging Interests," *IEEE Computer*, November 1990, pp. 51 - 57.

[7] Kerschberg, L., "The Role of Loose Coupling in Expert Database System Architectures," *Proceedings of the 5th International Conference on Data Engineering*, 1989, pp. 255 - 256.

[8] Kim, W., "Object-Oriented Databases: Definition and Research Directions," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 3, 1990, pp. 327 - 341.

[9] Mays, E., Lanka, S., Dionne, B., and Weida, R., "A Persistent Store for Large Shared Knowldege Bases," *Proceeding of the 6th International Conference on Data Engineering*, 1990, pp. 169 - 175.

[10] Sheth, A. and O'Hare, A., "The Architecture of BrAID: A Sytem for Bridging AI/DB Systems," *Proceedings of the 6th International Conference on Data Engineering*, 1991, pp. 570 - 581.