다사용자용 전문가 시스템 구축을 위한 Persistent Rule Object Management System

김 일 곤

경북대학교 자연과학대학 전자계산학과

요 약

요즈음 구축되는 지식 기반 시스템은 다목적, 다사용자에 의한 활용 필요성 때문에 지식베이스가 점점 커지고, 축적된 전문 지식을 공유할 필요성이 점점 커진다. 따라서, 이렇게 형성된 여러 개의 다양한 지식베이스를 지속적으로 관리할 수 있는 메카니즘이 필요하다. 이런 메카니즘의 구축에 객체 지향 데이타베이스 관리 시스템의 기술과 지식 기반 시스템의 기술이 통합 활용되어야 하기 때문에, 지식베이스의 확장과 응용 영역 확대에 따른 데이타, 규칙, 그리고 지식의 공유 및 지속성을 유지하는 한편, 이에 필요한 추론 방법의 변화를 분석하여 지식베이스가 분산되어 있더라도 이를 효율적으로 활용할 수 있는 추론 엔진을 설계하고, 여러 사용자가 필요한 지식 기반 시스템을 사용할 수 있도록 지식베이스의 공유를 위한 지속적 관리 시스템의 설계를 그 목적으로 한다.

I. 서론

새로이 구축되는 전문가 시스템의 지식베이스가 점점 커지고, 축적된 전문 지식을 공유할 필요성이 점점 커짐에 따라 여러 개의 다양한 지식베이스를 지속적으로 저장, 관리할 수 있는 메카니즘이 필요하다[4,9,11].

본 논문에서는 지식 기반 시스템의 기술에 객체 지향 데이타베이스 구현 기술을 접목시켜 지금까지 혼자 사용하는 형태로 구현되고 있는 각종 지식베이스를 여러 사용자가 공유할 수 있는 방법을 제시하고자 한다. 멀티미디어 기술의 발달로 인한 지식베이스의 공유가 꼭 필요하므로 객체 지향 데이타베이스 관리 시스템과 객체 지향 전문가 시스템 구축 언어[1,10], 두 시스템을 긴밀 결합한 지식베이스의 지속적 관리 시스템의 구현이 필수적이다[6,7,12]. 왜냐하면, 객체 지향 데이타베이스 관리 시스템과 객체 지향 전문가 시스템 구축 언어 모두 객체 지향 프로그래밍 기법에 기반을 두고 있기 때문에 데이타 구조가 일치하고, 표현력과 데이타 처리 능력면에서 상호 보완할 수 있기 때문이다. VLSI/CAD, CASE, AI분야에서 사용되는 데이타는 대부분 구조가 복잡하고 다양하며 데이타 간의 관계가 다양한 형태를 가지므로 이러한 데이타를 관계형 데이타베이스에서 요구되는 테이블 형태로 저장하여 처리하기 위해서는 변환(Conversion) 과정[5]이 복잡한데 비해,

객체 형태로 저장하게 되면 프로그래밍 언어에서 사용되는 데이타와 저장되는 형태가 유사하기 때문에 더 효율적이다. 또한, 객체 지향 데이타 모델은 일반성 관계 표현, 구성 관계의 표현, 메쏘드 사용등으로 인해 데이타가 가진 의미를 표현하는 능력이 훨씬 크다고 할 수 있다.

본 논문에서 활용하고자 하는 객체 지향 데이타베이스 구현 기술은 객체 지향 데이타베이스 관리 시스템의 관점에서 보면 아직 정형화되지 못했고, 보완해야 할 문제점을 많이 가지고 있지만, 지식베이스를 저장하고, 공유하는데 필요한 기술의 관점에서 보면 규모가 적고, 그 활용 가능성이 아주 높기 때문에 대규모 지식베이스의 공유를 위한 지속적 저장 시스템을 활용하고자 한다. 본 논문에서는 객체에 지속성(Persistency)을 주기 위한 방법으로 많은 연구가 이루어진 객체 식별자(Object IDentifier) 관리 기법, 객체 관리를 위한 버퍼 구현 기술, 객체의 주기억 장치 저장 구조와 보조 기억 장치 저장 구조를 변환하는 기술, 스키마 테이블 관리 기술[3]등을 활용하는데, 지금까지 열거한 기술은 객체 지향 데이타베이스 구현 기술중 가장 필수적인 기본 기술로써, 이미 그 개념이 정립되어 있고, 활용 영역에 따른 구조만 변형되고 있기 때문에 지식베이스 공유화에 필수적인 요소로 생각한다. 따라서 본 논문에서 제시하는 지식베이스를 공유하는 지속적 관리 시스템이 완성되면 지식 기반 시스템과 데이타베이스 관리 시스템의 통합 구현 기술 확보에 결정적 기여를 하게 되며, 지식 기반 시스템과 데이타베이스 관리 시스템 모두를 활용할 수 있기 때문에 많은 응용 시스템을 효과적으로 구축할 수 있다.

II. 객체 지향 규칙망

다사용자를 위한 지식베이스의 지속적 관리 시스템을 구축하기 위해서는 영역 지식과 프로그램 지식 모두 객체 지향 프로그래밍 언어를 이용하여 표현되어야 하기 때문에 추론 메카니즘 또한 객체 지향적으로 관리되어야 한다. 이를 위해 여러 개의 규칙 집합으로 구성된 규칙베이스와 여러 개의 사실 집합으로 이루어진 사실베이스를 한 개의 지식베이스로 구성하여 한 개의 지식베이스를 한 개의 데이타베이스에 저장한다. 이 때 초기 지식 및 컴파일된 추론 규칙망은 객체 식별자를 가리키는 구조로 변환하고, 데이타베이스에 저장된 객체 지향 규칙망을 다시 호출하여 사용하고자 할 때는 다시 객체 식별자가 가리키는 객체를 주기억 장치에서 포인터가 가리키는 구조로 변환하는 방법을 사용한다. 구현된 지식 기반 시스템이 여러 개의 지식베이스를 이용하고자 하는 경우에는 한 개의 지식베이스를 위해 만들어진 객체 지향 규칙망을 병합할 수 있는 방법도 제공해야 한다. 이 때, 주기억 장치에 이용하고자 하는 지식베이스가 없으면, 지식베이스 잘못(Knowledge-Base Fault)가 일어나고, 그 지식베이스는 객체 잘못(Object Fault)로 인식하여 객체 관리기에서 객체 버퍼에 지식베이스를 가져오게 된다. 지식 기반 시스템에서 사용하는 지식베이스에 대한 정보를 저장, 관리하기 위해 스키마 관리기를 이용하며, 지식 기반 시스템에서 제공하는 최상위 수준의 메뉴(Top Level Menu)에 따라, 하나의 스키마를 형성하게 된다. 이용하는 지식베이스의 버젼이 바뀌는 경우를 위해 버전 관리기가 필요하고, 사용자가 선택한 버전에 맞도록

해당 지식베이스를 호출하는 방법을 사용한다. 지식베이스의 지속적 저장 모델에서 제공하는 연산은 검색 연산과 수정 연산이 있는데, 검색 연산은 지식베이스에서 필요한 정보를 가져와서 활용하는 경우에 해당하고, 수정 연산은 현재 사용되고 있는 지식베이스는 그대로 두고 지식 공학자가 지식베이스의 수정을 원하는 경우에 이를 지원하고록 한다. 검색 연산은 지식베이스 전체 또는 지식베이스의 일부분을 보조 기억 장치로 부터 주기억장치의 작업 공간으로 읽어 오는 것을 의미한다. 지식베이스의 일부분을 읽어 올 때, 지식베이스를 구성하고 있는 계층 스키마 구조에서 하위 계층에 있는 모든 지식베이스는 읽어 오지만, 상위 계층에 있는 지식베이스는 읽어 오지 않는다. 지식베이스의 일부분을 보조 기억 장치로 부터 읽어 올 때, 하위 계층에 있는 모든 지식베이스의 연결 구조가 주기억 장치의 작업 공간에 똑같이 복사되어야 하기 때문에 Two Pass과정을 거친다. 처음 Pass에서는 지식베이스의 연결 구조를 따라 가면서 각 지식베이스에 해당하는 한 개의 노드가 작업 공간에 만들어진다. 처음 Pass에서 아직 만들어지지 않은 하위 구조의 지식베이스를 연결하는 포인터에 대한 정보를 관리 하기 위해 연결 정보 테이블을 만든다. 두번째 Pass에서는 각 노드를 연결하기 위해 연결 정보 테이블의 주소를 이용하여 실제로 각 지식베이스를 연결하여 주기억 장치의 작업 공간으로의 복사과정이 완료된다.
수정 연산은 지식베이스 전체 구조에서 각 부분 지식베이스를 수정, 첨가, 삭제하는 것을 의미하며, 이러한 작업은 개별적으로 Atomic하게 이루어진다. 수정 연산에서 충족시켜야 할 조건은 지식베이스의 계층 구조에서 Loop를 만들지 말아야 한다. 수정 연산에서 일치성 유지를 위하여 확인해야 할 지식베이스는 보조 기억 장치에 저장되어 있는 지식베이스이다. 작업 공간에서 수정 연산이 실행된 부분 지식베이스가 있고, 작업 공간에 있는 부분 지식베이스보다 상위에 있는 지식베이스를 검색하는 경우에는 수정 연산이 실행된 지식베이스를 이용하는 것이 아니라 보조 기억 장치에 저장되어 있는 공유된 지식베이스를 이용한다.

문제 영역을 분할하여 구성된 지식베이스가 공유된다는 것은 여러 명의 사용자가 필요한 때에 같은 지식베이스를 호출할 수도 있고, 수정 권한이 있는 지식 공학자가 지식베이스 수정 작업을 진행하고 있는 동안에도 다른 사용자는 지식베이스를 사용할 수 있고, 수정 작업이 완료되었을 때, 별도의 분배 작업을 거치지 않더라도 곧바로 사용자에게 전달되는 잇점이 있다. 하지만 이를 지원하기 위해서는 데이타베이스 기능 구현 기술, 추론 엔진 구현 기술에 관한 상당한 기술력을 필요로 한다.
지식베이스는 대개의 경우 사실과 규칙으로 구성되어 있기 때문에 지식베이스 공유 문제는 사실의 공유와 규칙의 공유로 분리하여 생각하여야 한다. 사실은 객체 지향 프로그래밍 기법을 이용하여 표현할 수 있고, 속성의 상속이나 계층적 구조 형성, 메쏘드의 활용과 같은 객체 지향 언어가 가진 특성을 잘 이용할 수 있다. 하지만 규칙의 표현 방법에 있어서는 전체적인 구조 형성은 계층적 구조를 형성할 수 있고, 메쏘드도 활용할 수 있지만, 속성 상속은 이용할 수 없다. 그렇다고 규칙의 표현에서 속성 상속을 이용할 필요성이 없기 때문에 이점은 별 다른 문제점으로 생각하지 않는다. 지식베이스가 분할 구성될 때, 사실베

이스는 크기가 그다지 크지 않기 때문에 분할할 필요성은 거의 없다. 그러나 규칙베이스의 경우는 규칙의 갯수가 점점 많아지면, 규칙 집합단위로 형성된 것중에서 필요한 것만 선택하는 방법을 지원해야 한다. 문제 영역을 몇 개로 나누어 필요한 지식을 제공하는 메뉴를 생각하면, 각 메뉴에 따라 규칙베이스 중 필요한 규칙 집합과 사실베이스를 묶어서 사용자에게 제공하는 방법이다. 이 때, 사실베이스와 규칙베이스는 보조 기억 장치에 저장되어 있고, 사용자가 필요로 하면 그것을 주기억 장치로 보내 주는 역할을 제공한다.

객체 지향 프로그래밍 개념에 바탕을 둔 전문가 시스템 구축 언어가 객체 서버 구조를 이용한 객체 지향 규칙망을 구성하기 위해 Rete Node Network[8]의 구조를 수정하여, 지식베이스간의 지식 참조 방법[1,2]과 지식 기반 시스템의 필요성을 만족시키는 엔진을 설계하고, Ruleclass로 나누어 지식베이스를 구성한다. 사용자의 지식베이스에 대한 다음 요구사항을 지원한다. 첫째, Super가 없는 규칙 집합을 요구하는 경우, 둘째 Super가 있는 규칙 집합을 요구하는 경우, 세째 한 개의 규칙 집합을 요구하는 경우, 네째 여러개의 규칙 집합을 요구하는 경우이다. 시스템에서 제공하는 메뉴가 있고, 각 메뉴에 의해 규칙 집합을 호출(call)한다. 호출된 규칙 집합에 대한 추론 네트웍을 사용자에게 전달한다. 초기 Fact에 의해 추론된 상태에서 필요한 입력을 기다리는 형태로 추론 네트웍을 전달한다.

III. 지속적 저장 시스템의 구조

객체 지향 데이타베이스 관리 시스템과 객체 지향 전문가 시스템 구축 언어를 긴밀 결합(Tightly Coupling)하는 것은 객체 지향 데이타베이스 관리 시스템이 가지는 특징중 불필요한 것들이 많기 때문에 규칙과 사실의 저장과 공유를 위해 지식의 지속적 저장 관리 시스템(Persistent Object Store Management System)을 구현하는 것이 효율적이다. 객체 지향 데이타베이스 관리 시스템은 여러 응용문제 영역에 적합할 수 있는 확장적이고, 여러가지 경우에 적용 가능한 특징들을 많이 가지고 있기 때문에, 지식의 지속적 저장 관리 시스템을 설계, 구현하는 것이 지식 기반 시스템이 가지는 특징에 부합된다.

본 시스템에서는 하나의 서버 기계에 전체 지식베이스가 저장되고, 여러 클라이언트 기계에서 이들 데이타를 접근할 수 있도록 지원한다.

각 클라이언트 기계에서 클라이언트 프로세스가 서버와의 연결을 요구하면, 서버 기계의 데몬(Daemon) 프로세스가 해당 클라이언트 프로세스를 위한 저장 서버 프로세스를 Fork해 준다. 그러므로, 클라이언트 프로세스와 저장 서버 프로세스는 1;1 대응 관계에 있으며, 이들 저장 서버 프로세스들은 공유 기억장소(Shared Memory)의 페이지 버퍼(Page Buffer)를 통하여 데이타들을 공유하면서 이들 공유 데이타에 대한 동시성 제어(Concurrency Control)는 공유 기억장소에 있는 잠금 테이블(Lock Table)을 이용하여 조정한다. 각 클라이언트 프로세스는 그 클라이언트 고유의 객체 버퍼(Object Buffer)를 가지고 있어, 해당 클라이언트에서 사용하는 객체에 대한 Cache 역할을 한다.

지식베이스의 지속적 관리 시스템 구조를 통해 제공되는 각 기능에 대한 그 중요성으로는 지식베이스의 지속성 유지, 지식베이스의 일치성 유지, 지식베이스의 병행성 유지가

있다. 지식베이스의 지속적 저장 모델에서 지식베이스는 보조 기억 장치에 저장되어 있고, 여러 명의 지식 공학자에 의해 공유된다. 각각의 지식 공학자는 별개의 작업공간(Workspace)을 할당받는다. 사용자가 지식베이스 A에서 필요로 하는 작업을 수행하고자 할 때는, 지식베이스 A는 공유되어 있기 때문에 지식베이스 A는 사용자의 작업 공간으로 Download되어야 한다. 작업 공간에 Download되어 있는 지식 베이스가 수정될 때는 특별한 작업 공간을 설정하여 수정하도록 하고, 다른 사용자에 의해 공유되지 않도록 한다. 지식 공학자는 지식베이스를 임의대로 수정할 수 있지만, 수정된 내용이 Commit될 때만 공유된 지식베이스를 변경하도록 한다. 또한 지식베이스의 지속적 저장 모델에서는 객체 식별자(Object IDentifier) 클래스를 지원해야 하고, 데이타베이스 스키마의 클래스에 대해 각각 하나의 객체 식별자 클래스가 존재하며, 스키마의 클래스가 가지는 IS_A 상속성 관계를 그대로 유지한다. 객체의 생성과 접근에 대한 연산을 제공하고, 지식베이스 객체 생성자는 새로 생성되는 객체를 객체 버퍼풀에 놓고 새로운 임시적인 객체 식별자를 할당받는다. 새로 생성되는 객체는 할당된 객체 식별자와 해당 객체 식별자 클래스의 메쏘드를 통해 지속적인 객체들과 동일하게 처리된다. 지식베이스 객체 액세스 메쏘드는 액세스하려는 객체를 객체 버퍼풀에서 찾는다. 만일 그 객체가 버퍼풀에 없다면, 지식베이스 포인터인 객체 식별자를 통해 지식베이스에서 그 객체를 버퍼풀에 가져온다. 그 다음에 그 객체에 대한 메모리 주소값을 반환한다. 따라서 메쏘드는 객체의 메모리 존재 여부와 객체의 지속성 여부에 상관없이 객체 잠금(Lock) 정보를 가지고 액세스한다. 포인터 스위즐링 기법에 의하여 디스크에 저장될 때는 메모리 포인터에 의한 참조(reference by memory pointer)를 객체 식별자에 의한 참조(reference by object identifier)로 바꾸고, 주기억장치에 가져올 때는 객체 식별자에 의한 참조(reference by object identifier)를 메모리 포인터에 의한 참조(reference by memory pointer)로 바꾼다. 객체 참조시 객체 버퍼 관리기를 통해 버퍼에 있는 경우에는 객체 식별자를 통해 참조하고, 버퍼에 없는 경우에는 객체 서버 저장 시스템에서 객체를 가져온다(getObject). 이 때, Lock 정보가 필요하다. 임시 식별자 테이블(Temporary OID Table)을 이용하여 새로 생성된 임시 OID들에 대한 정보를 유지한다. 객체 식별자를 클래스 별로 생성하기 위해 클래스 별로 생성된 임시 객체 시별자 갯수에 대한 정보를 유지하는 테이블이 필요하다.

IV. 지속적 저장 시스템의 구성 요소

지속적 저장 모델의 구성 요소로는 카탈로그 관리기, 객체 관리기, 지속적 이름 관리기, 스키마 관리기, 트랜잭션 관리기, 객체 저장 관리기, 버젼 관리기로 구성되어 있다[3]. 카탈로그 관리기는 클라이언트에서 사용자가 지식베이스를 요구하면 사용자의 자격 여부를 판단하고, 객체 관리기는 현재 사용하고 있는 지식베이스에서 다른 지식베이스가 필요할 때, 객체 지향 데이타베이스에 저장된 지식베이스를 가져오고, 보조 기억 장치에서 주기억 장치에 가져온 지식베이스의 구조를 바꾸는 역할을 수행한다. 지속적 이름 관리기는 지식베이스가 지속적으로 저장되고 공유될 수 있도록 지속성을 부여 받은 각 객체에 지속적 이

름을 부여한다. 스키마 관리기는 사실과 규칙으로 이루어진 지식베이스가 객체 지향 데이타모델로 관리되기 위한 정보를 관리한다. 트랜잭션 관리기는 공유 지식베이스에 대한 수정 권한이 있는 지식공학자가 지금 이용되고 있는 지식베이스와 독립적으로 수정 작업이 이루어질 수 있도록 지원하고, 각 사용자가 지식베이스를 사용한 결과를 저장하고자 할 때, 또는 먼저 저장해 놓은 것을 읽고자 할 때, 그 작업이 올바르게 수행될 수 있도록 지원한다.

카탈로그는 데이타베이스 관리 시스템의 전반적인 정보를 기록, 관리하는 장소이다. 이러한 정보에는 각 데이타베이스에 대한 정보, 시스템에 등록된 사용자와 그의 확인 정보, 시스템 내의 각 클래스 구조 정보, 정보의 종류별 갯수, 용량등이 포함될 수 있다. 이러한 정보들을 모두 카탈로그에서 관리하는 것은 시스템에 병목현상을 일으키게 하므로, 클래스의 명세는 클래스 SCHEMA의 인스턴스로 저장, 활용하도록 하고, 객체 식별자 부여를 위해 데이타베이스 식별자의 정보는 카탈로그에서 하지만, 클래스 식별자와 인스턴스 식별자는 클래스 OID에 의해 관리하도록 한다. 그러므로 카탈로그에서 관리하는 정보는 시스템 내에 등록되는 사용자에 관한 정보와 시스템 내의 데이타베이스의 생성, 소멸에 관한 정보이다.

객체 관리기에서 다루고 있는 객체는 여러 개의 객체가 모여 형성된 복합 객체(Complex Object)로서 객체 식별자를 가진다. 복합 객체는 기본 값(Atomic Value), 집합 생성자, 튜플 생성자의 임의의 조합으로 형성된다. 객체 관리기에서 제공하는 기본 연산들은 객체의 생성, 삭제, 수정, 전송의 기본 지식베이스 연산과 지식베이스 스키마에 대한 연산, 그리고 다양한 객체 복사, 동일성 검사등의 객체 지향 연산으로 구성된다. 객체 관리기는 객체 식별자 관리기, 객체 이름 관리기, 객체 버퍼 관리기를 갖고 있으며, 그 기능은 다음과 같다. 객체 식별자 관리기는 사용자에게 보이지 않는 객체 식별자 클래스로 구성된다. 객체에 대한 연산은 객체 식별자에 의해 이루어진다. 객체 식별자는 시스템에서 유일하며, 재사용되지 않는 식별자로서 객체에 대한 접근에 있어 근간을 이룬다.
객체 이름 관리기는 사용자에게 주어지는 NameTableManager 클래스로 구성된다. 객체에 대한 접근은 객체의 이름을 통해서 이루어질 수 있으며, 이는 객체의 이름을 객체 식별자로 변환함으로써 이루어진다. 객체 버퍼 관리기는 ObjectBuffer 클래스로 구성된다. 객체의 연산은 객체 버퍼 공간에서 이루어진다. 객체 버퍼는 객체 서버 저장 시스템과 객체의 전송을 담당하며, 객체 연산에 필요한 접근 체계를 제공한다. 객체 연산에 의한 객체의 상태 변화는 객체 버퍼 공간에서 이루어지게 된다. 객체 관리기의 서브시스템인 객체 버퍼 관리기는 이러한 객체 버퍼 공간을 관리한다. 객체 관리기에서의 객체는 메모리 저장 양식(Memory Format)을 가지며 객체 서버 저장 시스템의 객체는 디스크 저장 양식(Disk Format)을 가진다. 클라이어트 모듈로의 객체의 전송은 메모리 저장 양식에서 디스크 저장 양식으로의 활성화(Activation)가 동반된다. 객체 서버 저장 시스템의 객체의 전송은 디스크 저장 양식에서 메모리 저장 양식으로의 비활성화(Deactivation)가 동반된다. 객체 서버 저장 시스템에서의 객체들간의 참조 관계는 객체 식별자로 이루어지나, 클라이언트 모듈에서는

빠른 접근을 위해 메모리 포인터로 접근한다. 객체의 접근이 객체 식별자로 이루어질 때, 포인터 스위즐링 기법에 의해 해당 객체 식별자는 메모리 포인터로 변환된다. 객체의 비활성화시 메모리 포인터에서 해당 객체 식별자로의 변환이 이루어진다.

객체 버퍼 관리기는 ROD(Resident Object Descriptor), ROT(Resident Object Table entry), ObjectBuffer 구조로 이루어진다. ROD는 클라이언트의 ObjectBuffer에 상주하는 객체의 조회에 있어 사상 역할을 하는 구조로서, 데이타베이스 연산과 ObjectBuffer에서의 객체의 관리에 필요한 많은 정보를 유지한다. 객체 버퍼에 상주하는 객체는 하나의 ROD 구조를 가진다. ROT는 ObjectBuffer가 유지하는 테이블의 한 엔트리로서 주어진 객체의 ROD를 가리킨다. ObjectBuffer는 ROT를 엔트리로 하는 테이블로서, 해싱 기법에 의해 유지된다. 클라이언트 시스템에서 사용자에게 할당되는 객체 버퍼 관리기 모듈은 한정된 객체 공간을 관리하게 된다. 객체 버퍼 관리기는 할당된 객체 공간의 효율적인 활용을 위해 객체의 스워핑 등과 같은 연산을 수행한다. 객체 버퍼는 데이타베이스 관리 시스템의 작업 공간으로서의 의미를 지니며, 객체 버퍼 관리기는 객체 버퍼의 일반적인 공간 관리와 객체에 대한 메모리 공간에서의 접근 연산과 데이타베이스에 대한 기본적인 연산을 제공하고 있다.

객체 지향 데이타베이스 관리 시스템에서의 지속적인 이름이란 프로그램 상에서 데이타베이스 내의 한 객체에 부여되어 그 프로그램이 종료된 이후에도 존재하며, 이후에 이를 통해 해당 객체에 대한 접근을 가능하게 하는 이름을 의미한다. 따라서, 지속적인 이름은 사용자에 의한 객체별 색인의 역할을 할 수 있다.

지속적인 이름 공간은 시스템 내의 사상 정보이므로 색인 기법에 의해 탐색되어야 하고, 가능한한 분리되어야 시스템 성능 향상을 기할 수 있다. 또한 사용자간의 일관성을 위해 READ/WRITE의 제약으로, 한 사용자의 조작 연산이 다른 사용자에 의해서 의미있는 것인지 여부를 시스템이 판단할 수 있도록 해야만 한다.

스키마 관리기는 데이타 모델에 의해 저장되는 객체들에 대한 명세들의 모임인 지속성 스키마와 데이타베이스의 객체들이 응용 프로그램 내에서 사용될 때 필요한 정보를 제공하는 런-타임 스키마를 관리하는 부분이다.

스키마 지원 모듈은 스키마 정보를 관리하는 스키마 명세 해쉬 테이블과 런타임 스키마와 지속성 스키마를 이름으로 접근할 수 있도록 지원하는 스키마 이름 테이블, 임시 식별자를 생성하고 관리하기 위한 임시 식별자 테이블이 있다. 명세 정보는 응용 프로그램 전반에 걸쳐 자주 사용되는 정보이므로, 이를 객체 관리 버퍼에서 다른 객체들과 같이 관리하기 보다는 효율적인 관리를 위해 명세만을 위한 명세 테이블을 지원하고 있다. 스키마에 대한 정보는 OID뿐만이 아니라 이름을 통해서 가져오는 경우가 많다. 그러므로, 이름을 가지고 명세 또는 런-타임 스키마 정보를 접근할 수 있도록 하는 기능이 제공되어야 한다. 주기억장치 내에서 이를 지원하는 것이 스키마 이름 테이블이다. 기본적으로 새로 생성되는 객체는 클라이언트에서 생성되는 임시 식별자를 가지고 있다. 이후 트랜잭션이 완료되어 객체를 데이타베이스에 반영할 때, 클라이언트는 새로 생성된 객체들을 위한 실제 OID들을 서버로 부터 한꺼번에 할당받아 임시 OID를 가진 객체들에게 실제 OID를 할당해 주게 된다.

명세들도 마찬가지로 새로 생성될 때는 임시 OID를 가지게 된다. 이를 지원하기 위한 것이 임시 식별자 테이블이다. 임시 OID를 지원하기 위한 또 하나의 테이블은 클래스 별로 생성된 임시 OID의 갯수에 대한 정보를 유지하는 테이블이다. 이 테이블은 현재의 트랜잭션이 완료되면, 서버로부터 클래스 별로 필요한 실제 OID들을 배당 받는다. 서버로부터 클래스별로 실제 OID를 배당받아 오면, 실제로 임시 OID에 실제 OID를 배당하는 작업이 이루어진다.

지속적 관리 시스템은 여러 사용자들에 의해 공유되는 지식베이스에 대한 일관성을 유지할 수 있는 방법을 제공해야 한다. 또한 지식 기반 시스템의 사용자들이 다른 사용자들에 의한 지식베이스 접근을 인식하지 않고, 필요한 작업을 수행할 수 있도록 하여야 한다. 이러한 요구를 만족하기 위해 제공되는 것이 트랜잭션이다. 트랜잭션은 그 수행이 여러 사용자에 의해 공유되는 지식베이스에 대한 일관성을 유지하는 프로그램 단위이다. 지식 기반 시스템들에서는 지식베이스에 존재하는 자료에 대해 수행되는 연산들을 모두 트랜잭션 내에서만 수행이 가능하도록 함으로써 지식베이스 전체의 일관성 유지를 가능하게 한다. 이러한 트랜잭션의 제공은 지식 기반 시스템의 사용자가 다른 사용자의 지식 기반 시스템 사용과 무관하게 자신의 작업을 수행할 수 있도록 한다.

지식베이스에 대한 일관성을 유지하기 위해서 기존의 데이타베이스 관리 시스템들에서 제공하고 있는 트랜잭션 모델이 갖는 연속성과 원자성을 기반으로 한다. 연속성이란 여러 사용자에 의한 여러 트랜잭션들이 동시에 수행되는 경우, 그 결과가 이들 트랜잭션들에 대해 임의의 순서를 주어 수행한 결과와 같아지는 임의의 트랜잭션들의 순서가 존재하는 성질을 말한다. 원자성은 각 트랜잭션이 다른 트랜잭션에 의한 간섭없이 공유 자료에 접근할 수 있고, 트랜잭션이 정상적으로 완료되었다면, 이 트랜잭션에 의한 효과가 영구적으로 데이타베이스에 반영되고, 정상적으로 완료되지 않은 경우에는 전혀 영향을 받지 않은 상태로 데이타베이스가 유지되는 성질을 말한다. 응용 프로그래머에 의하여 beginTransaction과 endTransaction으로 묶인 연산들은 전체가 완전히 수행되거나 또는 전혀 수행되지 않아야 한다.

연속성과 원자성을 만족한다는 것은 여러 트랜잭션들이 각각 수행한 결과들을 트랜잭션의 수행이 완전히 완료되기 전에 다른 트랜잭션들에 의해 사용할 수 없으며, 또한 하나의 트랜잭션에 의해 수행되는 연산들은 이 트랜잭션이 정상적으로 완료되지 못하는 경우에는 이 트랜잭션에 의해 수행된 모든 연산들에 의한 데이타베이스에 대한 효과는 없어져야 한다는 것을 의미한다.

지식베이스의 사실과 규칙은 하나의 객체이고, 각 객체는 클래스의 인스턴스이다. 객체 저장 관리기는 이러한 클래스와 인스턴스들을 데이타베이스에 지속적으로 저장하고 필요에 따라 이를 적절히 접근할 수 있도록 하기 위한 객체의 저장, 클래스의 생성 여러 기능들을 수행한다. 데이타베이스 내에서 객체들은 그들의 값과 고유의 식별자를 갖고서 자신의 내부 구조를 정의하고 있는 클래스에 관련되어 저장된다. 모든 클래스는 자신의 클래스 명세에 대해 시스템 내에서 고유의 식별자를 가진다. 이들 클래스들의 클래스 명세를

관리하고 각 클래스 명세에 대해 사상 테이블을 생성하는 일을 일괄적으로 처리하는 스키마를 시스템 내에 정의한다. 클래스 SCHEMA는 클래스 객체의 저장소로서의 역할을 한다. 이 클래스는 해당 데이타베이스에 각 클래스가 생성될 때 이들의 클래스 명세를 인스턴스로 가지며, 생성된 클래스에 대해 고유의 객체 식별자를 부여한다.

클래스의 생성은 인스턴스들을 저장할 화일을 생성하고, 이 클래스에 속하는 인스턴스들에 대한 OID-RID 사상 정보를 저장할 화일을 생성하며, 인자로 전달된 명세 정보를 클래스 SCHEMA의 인스턴스로서 생성한다. 이 때 해당 클래스에 대한 메쏘드와 애트리뷰트에 대한 명세를 가지는 객체들이 생성되어 각각 클래스 METHOD와 클래스 ATTRIBUTE의 인스턴스로서 저장된다.

버젼 관리기는 동일한 객체에 대해 여러 대안들을 유지하거나, 객체의 변화 과정을 유지한다. 버젼 생성 가능성을 클래스의 특성으로 부여하면 지식 공학자가 클래스 생성시에 이 클래스의 인스턴스에 대해 버젼을 생성한다는 것을 명시한다. 버젼은 객체와 동일하게 취급되고, 동일한 객체에 대해 동시에 여러 사람의 수정 작업이 가능하도록 하고, 유도된 버젼 관계를 계층적으로 제공하고 사용 버젼의 명시 여부에 상관 없이 이를 지원하도록 한다. 특정 버젼을 명시하는 경우와 디폴트 번젼에 의해 접근하게 되는 두 가지 경우가 있다. 디폴트 버젼은 사용자가 특정 버젼을 명시하지 않는 경우 접근되는 버젼으로, 하나의 객체에 대해 가장 최근에 생성된 버젼을 의미한다.

V. 결론

데이타베이스 기술을 활용하여 지식베이스가 데이타베이스에 저장되어 지식의 공유 및 지속성 유지가 가능함으로 인해 전문가 시스템이 혼자서 사용하는 형태가 아닌 여러 사용자가 공동으로 사용하여 전문 지식을 공유할 수 있는 방법을 제공하고, 구축된 전문가 시스템의 유지, 관리가 잘 이루어질 수 있는 지식베이스의 지속적 관리 시스템을 설계, 제시하였다. 본 시스템은 다양한 지식베이스의 구축을 쉽게하고, 활용 영역을 확대할 뿐만아니라, 지식공학자들이 개발한 전문가 시스템의 관리 및 유지가 아주 편리하고, 사용자들이 전문가 시스템을 쉽게 사용하는데 도움을 준다.

지식베이스는 서로 협력 하에 있는 다수의 지식 공학자에 의해 공유되며, 여러 사람이 대규모 지식 기반 시스템의 개발에 참여할 수 있다. 지식 공학자들은 추론을 하면서 지식을 구성하여 응용 시스템을 개발한다. 지식베이스의 사실베이스와 규칙베이스는 클래스와 인스턴스로 이루어지며, 객체 지향 프로그래밍 기법에 의해 표현된다. 이들은 객체 지향 데이타베이스의 객체와 동일하기 때문에 객체 관리기에 의해 지속적으로 저장, 관리될 수 있어 응용시스템의 신뢰도 향상에 기여한다.

인공지능, CAD/CAM, CASE, Office Automation등에 데이타베이스의 활용이 확대되면서, 그 필요성에 의한 객체 지향 데이타베이스가 상품화됨에 따라 많은 응용 영역에서 객체 지향 데이타베이스를 활용하기 때문에, 실제 응용 영역에서의 적합성이 좋은 규칙에 기반을 객체 지향 규칙 기반 구축 언어와 결합된 본 시스템이 구현되면, 상품성과 응용 영역에 따른

적합성이 뛰어난 지식베이스가 공유되는 지속적 관리 시스템이 구축될 수 있기 때문에 멀티미디어, HyperText, CASE, 설계, 공장 자동화, 사무 자동화 등 지식과 데이타베이스를 모두 필요로 하는 모든 영역에서 광범위하게 사용될 것이다.

참고문헌

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

[2] 김 일 곤, 지식 기반 시스템의 지식베이스 형성 및 유지 기법, 박사학위 논문, 서울대학교, 1991b.

[3] 홍 은 지, 이 영 훈, 박 현 주, 이 주 홍, 전 용 석, 박 상 현, 김 영 길, 한국형 객체지향 데이타베이스 시스템 개발 기술보고서, 서울대학교 인공지능연구실, 1992.

[4] 민 미 경, 대규모 지식베이스의 운용: 지식 표현, 매칭, 저장 관리 기법, 박사학위 논문, 서울대학교, 1993

[5] Abarbanel, R. M., Tou, F. N., and Gillbert, V. P., "KEEconnection: A Bridge Between Databases and Knowledge Bases", Richer M. H.(Eds.) AI Tools and Techniques, Ablex, 1989.

[6] Andrews, T. and Harris C., "Combining Language and Database Advances in an Object-Oriented Development Environments", Proceedings of Object-Oriented Programmings: Systems, Languages, and Applications, pp. 430-440, 1987.

[7] Ballou, N., et al., "Coupling an Expert System Shell with an Object-oriented Database System", Journal of Object Oriented Programming, pp.12-21, 1988.

[8] Forgy, C. L., "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Problem", Journal of Artificial Intelligence, September 1982.

[9] Prerau, D. S., Gunderson, A.S., Reinke, R. E., and Adler M. R., "Maintainabilty Techniques in Developing Large Expert Systems", IEEE Expert, pp. 71 - 80, 1990.

[10] Yoo, S. I., Kim, I. K., Park, C. H., Chang, H. J., Kim, T. G., and Min, M. K., "HEXPERT: An Expert System Building Tool", Proceedings of the International Conference on Tools for Artificial Intelligence '91, IEEE Computer Society, California, November 10-13, pp. 510-511, 1991.

[11] Yoo, S. I. and Kim, I. K., "DIAS1: A Diagnosing System for Automobiles with Electronic Control Units", International Journal of Expert Systems with Applications, Vol. 4, pp. 69-78, 1992.

[12] Zaniolo, C., et al, "Object Oriented Database Systems and Knowledge Systems", Proceedings of the first International Workshop on Expert Database Systems, pp. 49-64, 1984.