정보자원사전에 대한 서술논리 표현과 관리

김창화*

First Order Predicate Logic Representation and Management for Information Resource Dictionary

Chang-Hwa Kim

〈요 약〉

인터넷 등의 컴퓨터 통신 네트워크의 발달로 인하여 분산된 정보자원의 공유를 통한 자원에 대한 재사용성의 필요성이 대두되었다. IRD(Information Resource Dictionary)는 조직 내에서 관련된 모든 정보에 대한 데이터가 논리적으로 중앙화된 정보저장소(repository)이다. IRD 내의 데이터는 다른 데이터를 기술하므로 이른바 메타 데이터라고 하기도 한다. IRD의 사전(dictionary) 요소는 정보자원의 종류, 정보자원의 의미, 정보자원의 논리적 구조, 정보자원의 위치, 그리고 정보자원의 접근방법 등을 기술한다.

FIPS ANSI의 IRDS는 이항 관계를 이용하여 무결성 제약조건을 표현하므로 제약조건 규칙의 표현과 일반적인 추론 규칙의 표현이 제한되어 있으며, 다양한 형태의 무결성 제약조건의 표현과 IRD와 관련된 여러 정보의 도출 또는 추론 및 관리에 관한 사항은 IRD 응용 고유의 문제로 간주하여 언급하고 있지 않다. 한편, FIPS IRDS는 사용자가 SQL 및 IRD에 대한 전문적 지식이 없이는 사용자 질의 작성이 어려운 점등에 대한 문제점을 안고 있다.

본 논문은 FIPS IRDS의 기본모델에서 정보자원 표현, 정보자원들간의 관계, 정보자원의 관리 정보 구분을 명확히 하기 위해 정보자원 모델을 *정보자원 표현요소*와 *정보자원 관리요소*의 두 부류로 나누어 구분하고, 각 부류에 대한 자격 질의(competency question)를 통하여 유추된 요소들을 FIPS ANSI IRDS 기본 모델의 스키마 기술 레벨과 스키마 레벨에 첨가함으로써 그 기본 모델을 확장한다. 그리고, FIPS ANSI IRDS가 제공하는 IRD 기술과 관리 기능을 그대로 포함하면서 앞에서 문제점으로 지적된 제약조건 표현과 추론규칙 표현을 위하여 확장된 기본 모델을 중심으로 각 레벨의 구성 요소들의 형식적 의미(formal semantics)와 레벨 내 혹은 레벨 구성요소들간의 관계성(relationship), 그리고 제약조건의 표현과 질의 추론 규칙들을 식별하여 FOPL(First Order Predicate Logic)로 표현한다. 또한, 본 논문은 FOPL로 표현된 predicate들과 규칙들을 구현하기 위하여 prolog로 변환하기 위한 이론적 방법론을 제시하고 정보자원 관리를 위한 기본 함수들과 스키마 진화(schema evolution)를 위한 방법론을 제안한다.

* 강릉대학교 컴퓨터과학과

1. 서 론

인터넷을 비롯한 네트워크 기반의 분산시스템이 일반화됨에 따라 분산 환경 하에서 이미 생성된 자원을 공유하거나 재사용하기 위하여 활발한 연구와 제품 개발이 진행 중에 있으며, 이러한 환경에서 자원의 공유와 재사용성을 자원하기 위해서는 자원에 대한 메타 정보가 필수적이다.

최근의 정보사전(Data Dictionary: DD)은 프로그램, 사용자, 하드웨어, 의사결정 모델 등에 대한 정보까지 포함하면서 그 범위 면에서 확장되어 왔다. 이러한 확장된 DD의 개념이 바로 정보자원사전 시스템(Information Resource Dictionary System: IRDS)이다. IRDS는 시스템 내에서 관련된 모든 정보에 대한 데이터가 논리적으로 중앙화된 정보저장소(repository)이며, IRDS 내의 데이터는 다른 데이터를 기술하므로 일종의 메타 데이터이다[11,19].

IRDS의 사전(dictionary) 요소는 무슨 정보자원이 존재하는지(정보자원의 종류), 그 정보자원이 무엇을 의미하는지(정보자원의 의미), 그 정보자원의 논리적 구조는 무엇인지(정보자원의 논리적 구조), 정보자원이 어디에 위치해 있고(정보자원의 위치) 또 그 접근 방법은 무엇인지(정보자원의 접근방법) 등을 기술한다[16]. 따라서, 자원의 개발과 관리, 그리고 이용에 필요한 제반 정보자원들을 IRDS 요소들을 이용하여 정의할 수 있고, 이미 정의된 정보자원을 공유 또는 재사용함으로써 새로운 시스템의 개발과 관리 등의 관련 분야에 이용될 수 있다.

현재 큰 이슈로 대두되고 있는 CORBA[17]나 DCOM[9]과 같은 시스템들은 분산 환경에서 자원공유 기능을 제공하기 위해 앞에서 언급한 메타 데이터로 구성된 정보저장소를 기반으로 하고 있다.

미국 NBS(National Bureau of Standard)의 Institute for Computer Science and Technology는 FIPS의 기초를 형성할 IRDS를 위한 명세를 개발해 왔다. 이 명세가 초안인 ANS(American National Standard) IRDS에 대한 기초로서 ANSC (American National Standards Committee) X3H4에 의해 채택되었다. 그러나, ANSI IRDS는 ER 모델을 기반으로 하는데, 이 표준이 갖는 여러 장점에도 불구하고 무결성 제약조건(integrity constraints) 표현의 한계, 관계성(relationship)과 관련된 cardinality 표현의 불가능, 다른 관계성을 포함하는 관계성 표현의 불가능, 3항(3-ary) 이상의 관계성 표현의 어려움 등이 문제점으로 지적되어 왔다[14,16].

FIPS ANSI의 IRDS는 이항 관계를 이용하여 무결성 제약조건을 표현하므로 제약조건과 규칙의 표현이 제한되어 있으며 다양한 형태의 무결성 제약조건의 표현과 IRD와 관련된 여러 정보의 도출 또는 추론에 관한 사항은 IRD 응용 고유의 문제로 취급함으로써 이에 대하여 언급하고 있지 않다. 한편, IRDS의 구현을 위해 RDBMS 환경하의 국제 표준인 SQL을 중심의 IRD의 표현과 무결성 제약조건 검사, 그리고 사용자 질의에 관한 연구가 일부 행해졌다. 이 연구들이 FIPS ANSI IRDS 표준의 활용과 표준 목적에 부합한다는 면에서 그 기여도가 크다고 하겠으나 무결성 제약조건을 표현하는데 제약이 있다는 점[10], 사용자가 SQL 및 IRD에 대한 전문적 지식이 없이는 사용자 질의 작성이 어려운 점(IRD 관리자가 질의 작성을 제공한다 하더라도 사용자의 질의 종류가 매우 다양하고 비 정형적이며 질의내용 면에서 예측 불가능하기 때문에 그 한계점을 갖는다)등에 대한 문제점을 안고 있다.

FOPL(First Order Predicate Logic)은 지식에 대한 표현이 데이터베이스 언어에 비해 매우 자연스럽고, 특히, 제약조건과 규칙의 표현에 매우 강하며, 질의 자체가 비절차성을 제공하므로 사용자에 의한 질의 표현이 매우 단순하고 수월하다. 또한 FOPL 시스템은 도출 연역에 의한 강한

추론기능을 제공한다[6,8]. 이미 정의된 clausal form 은 그 자체가 질의에 그대로 이용되므로 자원에 대한 질의를 행하는 사용자에게 이용상의 단순성과 재사용성을 제공한다.

이러한 배경 하에 본 연구는 FIPS ANSI IRDS가 제공하는 IRD 기술과 관리 기능을 그대로 포함하면서 앞에서 지적된 문제점들을 해결하기 위해 FIPS ANSI IRD 기본 모델을 확장하고, 제약 조건 표현과 추론 기능의 강화, 그리고 질의표현의 단순성을 위해 IRD 확장 모델에 대한 FOPL 표현과 관리에 대하여 언급한다.

본 논문은 제 2 절에서 현 미국표준으로 채택된 FIPS ANSI IRDS의 목적과 구성에 대하여 살펴본 후, 그 문제점에 대하여 논한다. 제 3 절에서는 FIPS IRDS의 기본모델에서 정보자원 표현, 정보자원들간의 관계, 정보자원의 관리 정보를 명확히 하기 위해 정보자원 모델을 정보자원 표현요소와 정보자원 관리요소의 두 부류로 나누어 구분하고, 각 부류에 대한 자격 질의(competency question)를 통하여 유추된 요소들을 FIPS ANSI IRDS 기본 모델의 스키마 기술 레벨과 스키마 레벨에 첨가함으로써 FIPS ANSI IRDS 기본 모델을 확장한다. 그리고, 제 4 절에서는 앞에서 문제점으로 지적된 자연스러운 제약조건 및 규칙 표현을 위하여 확장된 기본 모델을 중심으로 각 레벨의 구성 요소들의 형식적 의미(formal semantics)와 레벨 내 혹은 레벨 구성요소들간의 관계성(relationship), 그리고 제약조건과 추론 규칙들을 식별하여 FOPL로 표현한다. 제 5 절은 FOPL로 표현된 확장된 IRD의 predicate들과 규칙들을 prolog 언어로 구현하기 위하여 FOPL규칙을 prolog로 변환하기 위한 이론적 방법론을 제시하고 정보자원 관리를 위한 기본 함수들과 스키마 진화(schema evolution)를 위한 방법론을 제안한다. 마지막으로 제 6 절은 본 논문의 의의, 구현상의 문제점 및 해결 방안, 그리고 추후 연구방향 등을 간략히 언급함으로써 결론을 맺는다.

2. FIPS IRDS의 기본모델과 문제점

IRDS의 모델요소는 정보자원의 종류, 의미, 위치, 접근방법 등을 나타낸다. ER(Entity Relationship) 모델 중심으로 표현된 FIPS ANSI IRDS는 이러한 사항 외에 몇 가지 목적을 가지고 설계되었다. 이 절에서는 제3절의 FIPS IRDS 기본 모델의 확장에 대하여 논하기에 앞서 FIPS IRDS 기본모델과 이 모델의 문제점에 관하여 살펴본다. 이를 위하여 2.1절에서는 FIPS IRDS 설계 목적에 대하여 살펴보고, 2.2절에서는 FIPS IRDS의 구성에 대하여 알아본 후, 2.3절에서는 FIPS IRDS의 문제점에 대하여 논하기로 한다.

2.1 FIPS IRDS 설계 목적

FIPS IRDS 설계의 주요 목적은 다음의 세 가지로 요약될 수 있다[4].

IRDS 설계의 첫 번째 목적은 기존 자료사전 시스템(Data Dictionary System: DDS)이 갖는 주요 특성과 기능을 IRDS가 포함하도록 하는 것이다. DDS의 모든 주요 판매자들로 하여금 다양한 표준 초안을 검토하고 추천하도록 하였고 이러한 제안들이 결국 시스템-표준 스키마(system-standard schema)를 구성하는 기본(core) 사전 시스템으로 반영되었다.

두 번째 설계 목적은 어떠한 표준도 모든 사용자들의 요구사항을 만족할 수 없다는 사실을 인식하고 가능한 한 IRDS를 융통성 있도록 하는 것이었다. 그러므로, 시스템-표준 스키마는 IRDS에서 이용가능 한 개체(entities), 속성(attributes), 관계성(relationships)과 상응하는 요소들을 표현한다. 그러나, 이 표준은 개별 사용자나 판매자들이 다른 개체나 속성, 또는 관계성들을 첨가할 수 있도록 하였다. 더 많은 융통성을 제공하기 위해 FIPS IRDS는 특정 인터페이스나 임의의 모듈들이 IRDS내에 존재하도록 요구하지 않는다. 그 이유는 이들이 서로 독립적이 되도록

해서 사용자들이 IRDS에서 바라는 선택 사양들을 선택할 수 있도록 하기 위해서이다.

세 번째 주요 목적은 IRDS가 기술의 이식성(portability)과 광범위한 사용자 환경을 지원하는 것이다. 결국 이 목적은 비 경험자를 위한 메뉴 방식 판넬(menu-driven panel)과 노련한 경험이 있는 사용자를 위한 명령어 인터페이스(command language interface)에 관한 명세를 기술함으로써 반영되었다.

2.2 IRDS의 구성

FIPS IRDS 명세는 기본 시스템-표준 스키마(core system-standard schema)를 구성하는 개체-타입(entity-types), 속성-타입(attribute-types), 그리고 관계성-타입(relationship-types)을 포함한다. 이 기본 시스템은 정보사전시스템의 구현상 표준의 일부가 되도록 권장되고 있으며 필요시 부가적인 스키마 기술어(schema descriptors)를 첨가함으로써 확장이 가능하도록 되어 있다. 그림 1은 FIPS IRDS 기본 모델 예를 간략하게 나타내고 있다.

IRDS 아키텍처는 ER 모델에 기초하며 정보자원사전(Information Resource Dictionary: IRD)과 IRD 스키마로 구성된다. IRD는 IRD 스키마 개체-타입, 속성-타입, 관계성-타입에 대응하는 인스턴스(instance)인 개체, 속성, 관계성들로 구성된다. 또한, IRD 스키마는 IRD 기술 레벨(IRD description level)의 메타 개체(meta entities), 메타 관계성(meta relationships), 메타속성(meta attributes) 들의 인스턴스들로 구성된다.

IRDS의 구조는 개체가 노드가 되고 관계성이 노드들을 연결하는 아크가 되는 의미망(semantic network)과 유사하다. 모든 관계성은 이항 관계(binary relation)이며, 개체들은 서로 관련될 수 있다. 속성도 개체나 관계성에 연관될 수 있다. IRDS의 또 다른 주요 특성은 강하게 유형화(strongly typed)되어 있다는 것이다. 개체, 속성, 혹은 관계성의 각 인스턴스는 각각 개체-타입, 속성-타입, 그리고 관계성-타입의 인스턴스와 대응된다. 기본 모델을 구성하는 개체-타입, 속성-타입, 관계성 타입이 그림.2에서 제시되며 이에 대한 설명은 다음과 같다.

개체-타입 IRD 스키마는 데이터, 프로세스, 혹은 외부(external)로 범주화되는 12개의 개체-타입들을 포함한다. FILE, RECORD, ELEMENT, DOCUMENT들이 주요 개체-타입들이다. BIT-STRING, CHARACTER-STRING, FIXED-POINT, 그리고 FLOAT 또한 ELEMENTS의 특성을 나타내기 위해 사용되는 데이터 표현 개체-타입들이다. SYSTEM, PROGRAM, 그리고 MODULE

IRD 스키마 기술 층	개체-타입	관계성-타입	속성-타입
IRD 스키마 층	ELEMENT, RECORD, etc.	RECORD-CONTAINS-ELEMENT	DATE-ADDED, LENGTH, LOCATION, etc.
IRD 데이터 층	Soc-Sec-No, Empl-Record, etc.	Empl-Record-Contains-Soc-Sec-No	23Nov85, 12 (Char) Bldg A-Room 3
운영 데이터	555-23-6666(Employee record for Kirk)	Empl-Record for Kirk-CONTAINS-(555-23-6666)	속성들은 운영 데이터 베이스에서 나타나지 못한다.

그림 1: ANSI IRDS 기본 모델 아키텍처

은 시스템 개체-타입들을 나타내며 USER는 유일한 외부 개체-타입이다.

속성-타입 기본 시스템 표준 스키마에서 속성-타입은 정보 환경을 기술하기 위해 조직에서가장 필요로 하는 요소들로 선택되었다. 이 타입들은 개체와 관계성들에 대한 일반 문서화는 물론 감사보고 추적(audit trail) 정보를 제공하기 위해 선택되었다.

기본 모델은 ACCESS-NAME, DESCRIPTIVE-NAME, ALTERNATE-NAME을 이용하여 여러 가지의 다른 명칭들이 한 개체에 관련될 수 있도록 하였다. ACCESS-NAME은 사용자가 이용하기 위한 주요 명칭이며 수월한 사용을 위해 그 길이가 짧고 IRDS에서 유일해야 한다.

DESCRIPTIVE-NAME은 ACCESS-NAME의 간략성으로 인한 의미의 제약성을 보완하기 위

IRD 스키마 기술 레벨 : Entity-type(개체-타입), Relationship-type(관계성-타입), Attribute-type(속성-타입)

IRD 스키마 레벨 :

개체-타입 :

SYSTEM	FILE	BIT-STRING
PROGRAM	RECORD	CHARACTER-STRING
MODULE	ELEMENT	FIXED-POINT
USER	DOCUMENT	FLOAT

속성-타입 :

개체 관련 속성 타입 :

ACCESS-NAME	DURATION-VALUE
ADDED-BY	HIGH-OF-RANGE
ALLOWABLE-VALUE	LAST-MODIFICATION-DATE
ALTERNATE-NAME	LAST-MODIFIED-BY
CLASSIFICATION	LOCATION
CODE-LIST-LOCATION	LOW-OF-RANGE
COMMENTS	NUMBER-OF-LINES-OF-CODE
DATA-CLASS	NUMBER-OF-MODIFICATIONS
DATE-ADDED	NUMBER-OF-RECORDS
DESCRIPTION	RECORD-CATEGORY
DESCRIPTIVE-NAME	SECURITY
DOCUMENT-CATEGORY	SYSTEM
DURATION-TYPE	

관계성 관련 속성 타입 :

ACCESS-METHOD	FREQUENCY
RELATIVE-POSITION	

관계성-타입 :

CONTAINS	GOES-TO
PROCESSES	CALLS
RESPONSIBLE-FOR	DERIVED-FROM
RUN	REPRESENTED-AS

그림 2: ANSI IRDS 기본 시스템-표준모델 구성요소

해 더욱 상세하고 의미 있는 명칭이 개체에 부여되도록 한다. 이 명칭 또한 IRDS에서 유일해야 한다. ALTERNATE-NAME은 동의어 또는 일명 기능을 제공하며 다른 이름들이 동일 개체에 부여될 수 있다.

그림 2에서 제시된 모든 속성-타입들이 모든 개체-타입들에 적용되는 것은 아니다. 그 예로서 NUMBER-OF-LINES-OF-CODE는 PROGRAM과 MODULE 개체-타입들에만 적용된다. 그러나, ACCESS-NAME, ADDED-BY, CLASSIFICATION, COMMENTS, DATE-ADDED, DESCRIPTION, DESCRIPTIVE-NAME, ALTERNATE-NAME, LAST-MODIFICATION-DATE, LAST-MODIFIED-BY, NUMBER-OF-MODIFICATIONS 및 SECURITY 속성-타입들은 데이터 표현 개체-타입을 제외한 모두에 적용된다.

몇몇 속성-타입들은 개체-타입들과 다 대 일 관계성을 갖는다는 의미에서 다중(multiple) 속성들이다. 예를 들어, ALTERNATIVE-NAME은 관련된 임의의 특별 개체에 대해 여러 개의 다른 값을 가질 수 있다. 그러므로, 개체 'ZIP_CODE'는 ALTERNATE-NAME 속성 값으로서 'ZCODE', 'ZIP', 'ZIPCODE'들을 가질 수 있게 한다.

관계성-타입. IRD 스키마에 의해 제공되는 관계성-타입은 정보자원 환경에서 개체들 사이에서 작용하는 중요한 관련성을 나타내기 위해 고안되었다. 기본 시스템에서 모든 관계성-타입들은 이항관계이며 관계성에 관련된 개체-타입들에 따라 자체 선언적으로(예, SYSTEM-CONTAINS-PROGRAM) 명명된다.

그림 2에서 제시되는 제약조건들은 관계성-타입들에 참여할 수 있는 개체-타입들을 나타내며 관련 개체와 관계성 메타데이터에 적용될 무결성 제약조건(integrity constraints)을 정의한다. 관계성-타입 또한 관련된 속성-타입들을 가질 수 있다. 예를 들어, IRD 스키마는 관계성-타입들인 SYSTEM-PROCESSES-FILE, PROGRAM-PROCESSES-FILE, MODULE-PROCESSES-FILE에 대해 속성-타입 ACCESS-METHOD를 부여할 수 있다.

함수(functions)와 프로세스(processes) 기본 IRDS는 개체-타입, 속성-타입, 관계성-타입의 특별 인스턴스는 물론 이 타입들에 대한 서술, 조작, 제어를 지원해야 한다. 스키마 유지보수와 출력은 개체-타입과 관계성-타입을 기술하고 IRDS내에 존재하는 그 타입들에 대한 정보를 제시하는 기능을 포함한다. 즉, IRDS는 자체선언적(self-descriptive)이어야 한다.

IRDS 개체군, 유지보수, 그리고 출력은 정보자원 자체에 대한 데이터를 포함해서 실질적 개체와 관계성 인스턴스의 생성, 조작, 출력을 말한다. 그러므로, 스키마 유지보수는 개체-타입인 FILE, PROGRAM과 관계성-타입인 PROGRAM-PROCESSES-FILE의 기술과 관련되는 반면 IRDS 유지보수는 PROGRAM 개체인 EMPL_UPDATE, FILE개체인 EMPL_PROFILE, 그리고 관계성인 PROCESSES(EMPL_UPDATE, EMPL_PROFILE)에 대한 데이터의 입력과 관련된다.

중요한 IRDS 출력은 impact-of-change 보고서인데 이 보고서는 변경(change)으로 인해 영향을 받는 모든 개체들을 리스트화 한다. 다른 출력 명세들은 개체와 관계성정보에 대한 특별한 보고서 양식을 포함한다.

2.3 FIPS IRDS의 문제점

앞 절에서 설명한 IRDS의 레벨화 및 각 레벨간의 추상화 관계, 그리고 각 레벨의 구성요소들은 자원에 대한 표현(정보자원 표현요소)뿐만 아니라 정보자원 자체에 대한 메타 정보(정보자원 관리요소)의 표현을 가능하게 해준다. 그러나, 몇몇 문헌들은 FIPS IRDS 기본 시스템-표준 스키마에서 관계성이 이항 관계로 표현됨에서 비롯되는 문제점들을 포함하는 다음과 같은 한계점과 문제점들을 지적해 왔다[10].

- FIPS IRDS는 관계성 cardinality의 표현을 지원하지 못하며 사용자가 개인고유의 모델링 제약조건을 첨가할 수 있도록 하는 기능을 제공하지 못한다. 완전한 cardinality를 지원하지 못한다는 사실은 데이터 모델자에게 모든 논리적 설계 단계에서 인정할 수 없는 부담을 안길 수 있다.
- FIPS IRDS의 기본 시스템-표준 스키마에서 관계성이 이항 관계로 표현된다. 이 모델에서 무결성 제약조건은 관계성으로 표현되는데, 결국 무결성 제약 조건도 이항 관계성으로 표현되기 때문에 사용자가 모델링 고유의 다양한 제약조건의 표현이 부족하다. 따라서 무결성 제약조건 표현을 위한 새로운 메커니즘이 도입되어야 한다.
- 이항 관계성만 지원하기 때문에 3항 이상의 관계성을 표현하기 위해 이를 이항 관계성으로 표현해야 하므로 정보 손실과 같은 문제점이 야기될 수 있다.

3. IRD 확장 모델

FIPS IRDS의 기본모델에서 정보자원 표현, 정보자원들간의 관계, 정보자원의 관리 정보를 명확히 하기 위해 3.1절에서는 정보자원 모델을 정보자원 표현요소와 정보자원 관리요소의 두 부류로 나누어 구분하고, 각 부류에 대한 자격요건을 자격질의(competency question)를 통하여 유추한다. 이렇게 유추된 요소들을 FIPS ANSI IRDS 기본 모델의 스키마 기술 레벨과 스키마 레벨에 첨가함으로써 확장된 모델을 3.2절에서 제안한다.

3.1 정보자원 표현과 관리를 위한 모델 기본 요소

일반적으로 자원은 보편적으로 물리적, 또는 논리적 객체 또는 추상적 객체 또는 activity가 될 수 있다. 자원은 주어진 activity를 수행하기 위해 요구되는 대상으로 정의된다. 자원은 activity 수행을 위해 그 자체의 역할을 행한다. activity를 위한 역할의 수행은 자원에 대한 연산(이 또한 activity)에 의해 수행될 수 있다. 따라서, 자원은 그 자체의 특성과 다른 자원 또는 다른 객체 및 activity와의 관련성, 그리고 그 역할에 의해 표현된다[7]. 이러한 성질에 있어 정보자원 또한 예외가 될 수는 없다. 이러한 성질은 자원을 나타내고 식별해 주는 주요요소들로서 자원에 대한 *정보자원 표현요소*로서 정의된다.

한편, 정보자원은 생성되고 라이프 타임을 통하여 변경되며 사용자의 요구에 의해 프로세스를 통하여 접근됨으로써 이용된다. 이와 같이 정보자원의 생성, 변경, 이용 등과 관련된 정보자원 요소를 *정보자원 관리요소*로 정의된다. 정보자원 표현요소는 자원을 표현하고 나타내는 요소인 반면, 정보자원 관리요소는 정보자원에 관련된 메타 정보로서 정보자원의 생성 때부터 라이프 타임 동안 생성, 소멸, 변경, 이용되는데 수반하는 정보를 표현하는 요소들이다.

이러한 정보자원 관리요소와 정보자원 표현요소들을 식별하여 모델 요소에 포함시킴으로써 기본 모델을 확장함으로써 FIPS IRDS 정보자원의 표현 기능을 확장시키고 정보자원의 구분을 명확히 하는 것이 본 절의 목적이다.

정보자원의 표현요소와 관리요소들을 다음과 같은 자격 질의(competency question)에 대한 해답을 구하는 과정에서 식별 가능하다.

정보자원 관리요소 식별을 위한 자격 질의

1. 정보자원 생성
   - 정보자원 IR은 누구에 의해 언제 IRD 내부로 생성되었는가 ?

2. 정보자원의 보안
- 정보자원 IR를 생성할 수 있는(또는 생성한) 사람(그룹)은 누구인가 ?
- 정보자원 IR를 변경할 수 있는(또는 변경한) 사람(그룹)은 누구인가 ?
- 정보자원 IR를 삭제할 수 있는(또는 삭제한) 사람(그룹)은 누구인가 ?
- 정보자원 IR를 검색할 수 있는(또는 검색한) 사람(그룹)은 누구인가 ?
- 정보자원 IR를 접근할 수 있는(또는 접근한) 사람 또는 그룹은 누구인가 ?

3. 정보자원의 변경
- 정보자원 IR은 누구에 의해 언제 변경되었는가 ?
- 정보자원 IR의 변경에 의해 영향을 받는 관련 정보자원들은 무엇인가 ?

4. 버전 제어
- 정보자원 IR 과 관련된 working set은 무엇인가 ?
- 정보자원 IR의 버전들은 무엇인가 ?

5. 정보자원의 접근 방법
- 정보자원 IR과 관련된 질의들은 무엇인가 ? 또한 그 질의 형식(명령어, 파라메터, 파라메터 타입 등)은 무엇인가 ?

자원에 대한 정보자원 표현요소 식별을 위한 자격 질의

1. 자원과 activity에서의 역할과의 관계
- activity 수행에 요구되는 자원은 무엇이며 또 그 역할은 무엇인가 ?
- activity 수행을 위한 자원 R의 역할을 수행하기 위해 요구되는 operation은 무엇인가 ?
- 주어진 자원 R은 어느 activity에 이용될 수 있는가 ?

2. 자원의 구조
- 자원 R을 구성하는 구성요소들은 무엇이며 또 어떻게 구성되어 있는가 ?

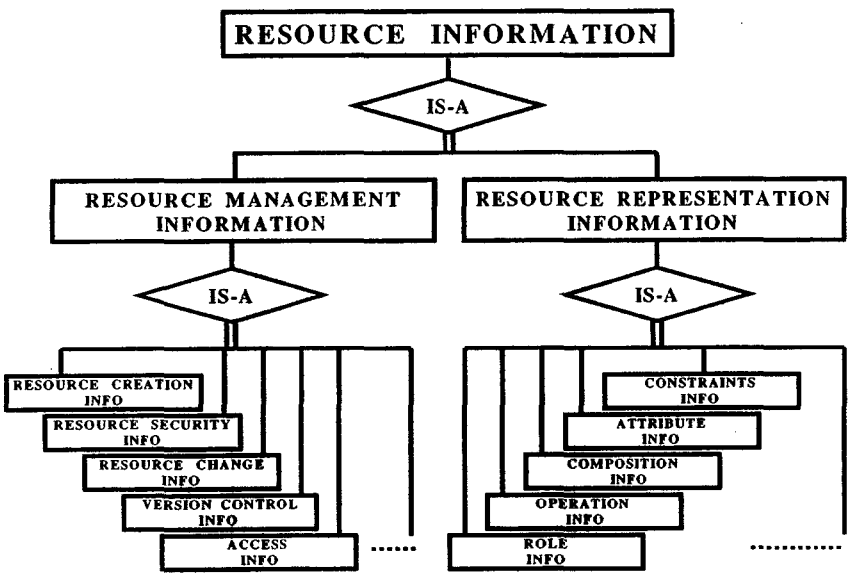

그림 3: 자격 질의로부터 유추된 IRD의 구성 체계 EA 모델

3. 자원의 속성
- 자원 R의 속성은 무엇이며 그 도메인은 무엇인가 ?
- 자원 R의 속성 A에 대한 속성 값은 무엇인가 ?
- activity 수행을 위해 고려되는 속성들은 무엇인가 ?

4. 자원들간의 관련성
- 자원 R은 다른 자원들과 어떠한 관련성을 갖고 있는가 ?

5. 제약조건
- 자원의 속성, 연산, 자원자체 혹은 다른 자원들과의 관계상에 주어진 제약 조건은 무엇인가 ?

위의 자격 질의의 해답을 구하는 과정에서 유추되는 정보자원 요소들을 중심으로 IRD의 분류체계를 EA(Entry Aspect) 모델[3,20](EA 모델은 실세계의 개념 또는 개체를 측면별로 분류하여 나타내는 모델이다)로 나타내면 그림 3과 같다. EA 모델에서 사각형은 실세계의 개체를 나타내는데 그림.3에서는 정보자원 요소를 나타낸다. 사각형 안의 이름은 정보요소의 명칭이다. IS-A를 포함하는 다이아몬드는 상위 정보요소와 하위 정보요소간의 일반화(generalization) 관계를 나타내는 측면이다. 그림 3을 좀 더 자세히 설명하면 다음과 같다. 정보자원의 정보(RESOURCE INFORMATION)는 정보자원 관리요소(RESOURCE MANAGEMENT INFORMATION)와 정보자원 표현요소(RESOURCE REPRESENTATION INFORMAT-ION)로 분류된다. 정보자원 관리요소는 정보자원 생성정보(RESOURCE CREATION INFO), 보안정보(RESOURCE SECURITY INFO), 변경정보(RESOURCE CHANGE INFO), 버전제어정보(VERSION CONTROL INFO), 접근 정보(ACCESS INFO) 등으로 분류되며, 정보자원 표현요소는 역할 정보(ROLE INFO), 역할에 대한 연산 정보(OPERATION INFO), 구성 정보(COMPOSITION INFO), 속성 정보(ATTRIBUTE INFO), 관계성 정보(RELATION INFO), 제약조건 정보(CONSTRAINTS INFO) 등으로 분류된다. 이렇게 분류된 정보 요소들은 3.2절의 확장 모델로 반영된다.

3.2 IRD 확장 모델

3.1절에서 제시된 정보 요소들을 중심으로 ANSI FIPS IRDS의 표준 기본 모델을 확장하여 그림.4에 제시하였다. 그림.4의 확장 모델 요소들 중 밑줄친 부분은 새로 도입된 요소들이며 밑줄이 없는 요소들은 ANSI FIPS IRDS 기본모델 구성요소로서 그 의미는 ANSI 표준에서 제시된 내용과 동일하다[4,5,10]. IRD 스키마 기술 층(IRD schema description layer), IRD 스키마 층(IRD schema layer), IRD 데이터 층(IRD data layer), 연산 데이터(operational data) 등의 4 계층을 기본 층으로 하여 구성된 ANSI 기본 모델은 상위 층의 구성요소와 하위 층의 구성요소들 간에 클래스와 인스턴스의 관계(또는, 상위 클래스와 하위 클래스들의 관계)를 갖는다. 확장된 IRD 기본 모델은 그림 4와 같이 기본적으로 두 층과 관련되어 있으나 자원에 대한 지식정보 표현을 위해 엄격하게 제한하지 않는다. 그 이유는 IRD를 구성하는 IRD 데이터 층의 ELEMENT 구성 요소는 IRD 표현을 위한 정보자원이며 따라서 이 구성요소는 운영 데이터로서도 이용가능하기 때문이다.

확장된 IRD의 IRD 스키마 기술 층(IRD schema description layer)은 ANSI 기본 모델의 개체-타입(entity-type), 관계성-타입(relationship-type), 속성-타입(attribute-type)외에 정보자원을 표현하기 위한 측면을 포함하는 측면-타입(aspect-type)으로 구성된다.

IRD 스키마 층에서 개체-타입에 소속된 인스

IRD 스키마 기술 레벨 : Entity-type(개체-타입), Aspect-type(측면-타입),
Relationship-type(관계성-타입), Attribute-type(속성-타입)

IRD 스키마 레벨 :

개체-타입 :

SYSTEM	FILE	BIT-STRING
PROGRAM	RECORD	CHARACTER-STRING
MODULE	ELEMENT	FIXED-POINT
USER	DOCUMENT	FLOAT
RESOURCE	CLASS	INSTANCE
OBJECT	ATTRIBUTE	RELATION
ACTIVITY	ROLE	OPERATION
METHOD	PARAMETER	
IRD	RESOURCE-INFO	
RESOURCE-MANAGEMENT-INFO	RESOURCE-REPRESENTATION-INFO	
RESOURCE-CREATION-INFO	RESOURCE-SECURITY-INFO	
CHANGE-INFO	VERSION-CONTROL-INFO	
ROLE-INFO	COMPOSITION-INFO	
ATTRIBUTE-INFO	RELATIONSHIP-INFO	
CONSTRAINTS-INFO	OPERATION-INFO	

측면-타입 :

RESOURCE-INFO-ON	IS-A
SUBCLASS-OF	INSTANCE-OF
A-PART-OF	HAS-ROLE
HAS-OPERATION	HAS-ATTRIBUTE
HAS-ATTRIBUTE-VALUE	HAS-RELATION
HAS-CONSTRAINTS	

속성-타입 :

개체 관련 속성 타입 :

ACCESS-NAME	DURATION-VALUE
ADDED-BY	HIGH-OF-RANGE
ALLOWABLE-VALUE	LAST-MODIFICATION-DATE
ALTERNATE-NAME	LAST-MODIFIED-BY
CLASSIFICATION	LOCATION
CODE-LIST-LOCATION	LOW-OF-RANGE
COMMENTS	NUMBER-OF-LINES-OF-CODE
DATA-CLASS	NUMBER-OF-MODIFICATIONS
DATE-ADDED	NUMBER-OF-RECORDS
DESCRIPTION	RECORD-CATEGORY
DESCRIPTIVE-NAME	SECURITY
DOCUMENT-CATEGORY	SYSTEM
DURATION-TYPE	

관계성 관련 속성 타입 :

ACCESS-METHOD	FREQUENCY	RELATIVE-POSITION

관계성-타입 :

CONTAINS	GOES-TO
PROCESSES	CALLS
RESPONSIBLE-FOR	DERIVED-FROM
RUN	REPRESENTED-AS

그림 4: IRD 확장 모델

턴스들은 ANSI 기본 모델의 스키마 층을 구성하는 구성요소들 외에 RESOURCE, ACTIVITY, ROLE, OPERATION, ATTRIBUTE, METHOD, PARAMETER들이 첨가되어 확장되었다. RESOURCE는 자원 클래스를 나타내며, ACTIVITY는 자원에 관련된 activity 클래스를 나타낸다. 또한 ROLE은 자원이 관련 activity에서 수행하는 역할에 관한 클래스를, OPERATION은 자원이 관련 activity에서의 역할 수행을 위해 요구되는 연산(operation) 클래스를, ATTRIBUTE는 자원의 속성 클래스를 각각 나타낸다. METHOD는 객체지향 모델에서 연산 수행의 상세 절차를 나타내는 메소드 클래스를 의미한다. PARAMETER는 연산 수행에 요구되는 또는 연산 수행에 의해 만들어지는 자원(소프트웨어 공학에서 입/출력 매개 변수를 의미)을 각각 나타낸다. 이외에 정보자원의 종류와 구성을 나타내기 위해 이용되는 인스턴스들은 앞 절의 자격 질의를 통하여 구분되었으며 그 종류와 의미는 다음과 같다.

IRD: 정보자원 사전 클래스
RESOURCE-INFO: 정보자원 클래스
RESOURCE-MANAGEM리 정보 클래스
ENT-INFO: 자원 관RESOURCE-CREATION-INFO: 자원 생성 정보 클래스(자원 생성: IRD에 정보자원의 삽입을 의미함)
RESOURCE-SECURITY-INFO: 자원 보안 정보 클래스
CHANGE-INFO: 자원 변경 정보 클래스
VERSION-CONTROL-INFO: 버전 제어 정보 클래스
RESOURCE-REPRESENTATION-INFO: 자원 표현 정보 클래스
ROLE-INFO: 역할 정보 클래스
COMPOSITION-INFO: 구성 정보 클래스
ATTRIBUTE-INFO: 속성 정보 클래스
RELATIONSHIP-INFO: 관계성 정보 클래스
CONSTRAINTS-INFO: 제약조건 정보 클래스
OPERATION-INFO: 연산정보 클래스

한편, 확장된 IRD 스키마 층에서 ANSI 기본 모델 외에 첨가된 측면-타입(aspect-type)의 인스턴스들은 측면들을 도입하여 확장한 구성요소들로서 정보자원 관리요소 및 정보자원 표현요소와 구성규칙과 제약조건을 표현하는 규칙들을 나타내는 정보요소들로서, 일반적 지식추론을 위해 사용자 질의 시 이용 가능한 predicate 기본 요소들이기도 하다. 측면-타입에 포함되는 인스턴스들과 그 의미는 다음과 같다.

RESOURCE-INFO-ON: 자원과 정보자원간의 관련성을 나타내는 측면
IS-A: 자원의 generalization 즉, subclass_of 또는 instance_of관계를 나타내는 측면
SUBCLASS-OF: 상위 클래스와 하위 클래스간의 관계를 나타내는 측면
INSTANCE-OF: 자원의 클래스와 인스턴스 관계를 나타내는 측면
A-PART-OF: 자원들 간의 구성요소 관계를 나타내는 측면
HAS-ROLE: 자원이 관련 activity에서 수행하는 역할 측면
HAS-OPERATION: 자원이 activity에서의 역할 수행을 위한 연산 측면
HAS-ATTRIBUTE: 자원이 갖고 있는 속성 측면
HAS-ATTRIBUTE-VALUE: 자원이 갖는 특정 속성의 값을 표현하는 측면
HAS-RELATION: 자원이 갖는 관련성을 표현하는 측면
HAS-CONSTRAINTS: 제약조건을 나타내는 측면

위의 측면들은 자원 표현 정보, 자원 관리 정보, 그리고 자원들간의 관계성 정보와 규칙들을 구성하고 질의에 이용되는 predicate들로 변환된다. 이 내용이 제 4 절에서 논의된다.

4. IRD 확장 모델의 FOPL 표현

제4절에서는 제3절의 IRD 확장모델에 포함되어 있는 개념을 atomic formula 형태로 정의하고 이 모델에 관련된 규칙들을 식별하여 FOPL로 지식 표현하는 과정과 내용에 관하여 언급한다. 이를 위해 4.1에서는 3.2절의 확장모델 각 요소에 대한 속성들과 요소들간에 존재하는 관계성을 중심으로 가장 단순한 개념인 atomic formula들을 정의하고, 확장모델에 내재된 제약조건과 추론규칙을 atomic formula들을 이용하여 FOPL 규칙들로 표현한다. 4.2절은 3.2절에서 제안한 확장모델과 4.1절의 atomic formula들 및 규칙들을 기반으로 정보자원 구성과 관련된 사실들(facts)들 표현하고. 정보자원 생성, 정보자원 변경에 관련된 제약조건 규칙들을 식별하여 FOPL로 표현하는 내용에 대하여 논한다. FIPS ANSI IRDS 기본모델은 ER 모델을 기반으로 한 반면, 확장모델은 객체지향 모델을 기반으로 하고 있기 때문에 확장모델로 표현된 자원들 사이에는 일반화(generalization) 관계에 의한 상속(inheritance) 관계가 존재할 수 있다. 이러한 상속관계에 대한 규칙이 4.3절에서 FOPL로 표현된다.

4.1 atomic formula의 정의와 semantics rules

3.2절에서 제시한 측면 요소들을 이용하여 IRD 구성을 위해 사용되는 atomic formula(즉, predicate)들과 이에 부여된 의미(semantics)들은 다음과 같다:

AF1: is_a(i,a,b): a가 b의 부분 클래스 혹은 a가 b의 인스턴스이다.

AF2: subclass_of(i,a,b): a는 b의 부분 클래스이다.

AF3: instance_of(i,a,b): a는 클래스 b의 인스턴스이다

AF4: resource_info_on(i,r): 정보 i는 자원 r에 관한 정보이다.

AF5: a_part_of(i,a,b): a는 b의 부분(구성요소)이다.

AF6: has_role(i,r,a,role): 자원 r은 activity a에서 역할 role을 수행한다.

AF7: has_operation(i,r,a,role,o): 자원 r은 activity a에서의 역할 role을 수행하기 위해 연산 o를 수행한다.

AF8: has_attribute(i,r,a): 자원 r은 속성 a를 갖는다.

AF9: has_attribute_value(i,r,a,v): 자원 r의 속성 a의 값은 v이다.

AF10: has_relation(i,r,rel): 자원 r은 관계 rel을 갖는다.

AF11: has_constraints(i,r,c): 자원 r은 제약조건 c를 갖는다.

위의 atomic formula들은 정보자원 표현을 위한 기본단위들이며 이들의 첫 번째 변수(혹은 상수) i는 위의 각 atomic formula 들이 정보임을 나타내는 정보자원 식별자로서 이들 정보자원을 관리 정보와 관련시키는 역할을 한다. 또한 이 식별자의 역할은 권한을 갖는 다수 사용자들이 단위 정보를 관리할 수 있도록 허락한다. 이 식별자는 사용자가 일일이 정의하고 입력하기에는 너무 번거롭고 IRD내에서 중복 정의될 수 있기 때문에 정보 식별자 정의 메커니즘을 도입한 시스템에 의해 자동 생성이 가능하다.

위의 기본 측면들 중 is-a 측면을 이용하여 그림.4의 IRD 확장 모델의 구성요소들의 관계가 그림.5에서 제시된다.

위의 atomic formula들 중 AF1, AF2, AF5의 predicate인 is_a와 a_part_of, 그리고 subclass_of는 전이관계(transitive relation)를 가지며 다음의 규칙들로 표현된다.

Rule1: $\forall a \forall c(\exists i1 \exists i2 \exists b(is\_a(i1,a,b) \wedge is\_a(i2,b,c)) \Rightarrow is\_a(\_,a,c))$

Rule2: ∀a∀c(∃i1∃i2∃b(a_part_of(i1,a,b)∧a_part_of(i2,b,c)) ⇒ a_part_of(_,a,c))

Rule3: ∀a∀c(∃i1∃i2∃b(subclass_of(i1,a,b)∧subclass_of(i2,b,c)) ⇒ subclass_of(_a,c))

Rule1과 Rule2에서 전이관계에 의해 추론된 결론 정보의 정보식별자는 시스템에 의해 정의될 수 있지만 정보 식별을 위해 IRD내에 존재하는 것이 아니라 추론에 의한 결과이므로 정보 식별자가 정의될 필요가 없다. 이러한 이유 때문에 추론된 결과 formula의 첫 변수(혹은 상수)는 '_'으로 표시하였고 이는 추론된 정보임을 나타낸다.

앞에서 정의된 atomic formula의 모든 변수와 상수들의 타입은 기본적으로 ACCESS-NAME이며, resource_info_on(i,r)에서 i와 r의 타입은 각각 RESOURCE-INFO와 RESOURCE이다. 이를 반영한 내용이 Rule4, Rule5, Rule6, Rule7, Rule8에서 정의된다.

Rule4: ∀ri∀r((resource_info_on(ri,r)⇒is_a(_,a,'ACCESS-NAME')∧is_a(_,b,'ACCESS-NAME') ∧ is_a(_,ri,'RESOURCE-INFO') ∧is_a(_,r,'RESOURCE'))

Rule5: ∀a∀b(∃i(is_a(i,a,b)) ⇒ is_a(_,a,'ACCESS-NAME')∧is_a(_,b, 'ACCESS-NAME'))

Rule6: ∀a∀b(∃i(subclass_of(i,a,b))⇒ is_a(_,a,'ACCESS-NAME')∧is_a(_,b,'ACCESS-NAME'))

Rule7: ∀a∀b(∃i(instance_of(i,a,b))⇒ instance_of(i,a,'ACCESS-NAME')∧instance_of(i,b,'ACCESS-NAME'))

Rule8: ∀a∀b(∃i(a_part_of(i,a,b))⇒ is_a(_,a,'ACCESS -NAME')∧is_a(_,b, 'ACCESS-NAME'))

has_role(i,r,a,role)에서 r, a, role은 각각 RESOURCE, ACTIVITY, ROLE 타입을 갖는다(Rule9).

Rule9: ∀i∀r∀a∀role(has_role(i,r,a,role)⇒is_a(i,r,'ACCESS-NAME') ∧ is_a(_,a,'ACCESS-NAME') ∧ is_a(_,role,'ACCESS-NAME') ∧ is_a(_,r,'RESOURCE') ∧ is_a(_,a,'ACTIVITY') ∧ is_a(_,role,'ROLE'))

has_operation(i,r,a,role,o)에서 r, a, role, o는 각각 RESOURCE, ACTIVITY, ROLE, OPERATION 타입을 갖는다(Rule10).

Rule10: ∀r∀a∀role∀o(∃i(has_operation(i,r,a,role,o)) ⇒ is_a(_,r,'ACCESS-NAME')∧is_a(_,a,'ACCESS-NAME')∧is_a(_,role,'ACCESS-NAME')∧is_a(_,o,'ACCESS-NAME')∧is_a(_,r,'ACCESS-NAME') ∧is_a(_,r,'RESOURCE')∧is_a(_,a,'ACTIVITY')∧is_a(_,role,'ROLE')∧is_a(_,o,'OPERATION'))

has_attribute(_,r,a)에서 r, a는 각각 RESOURCE, ATTRIBUTE 타입이다(Rule11).

Rule11: ∀r∀a(∃i(has_attribute(i,r,a)) ⇒ is_a(_,r,'ACCESS-NAME')∧is_a(_,a,'ACCESS-NAME')∧is_a(_,r,'RESOURCE')∧is_a(_,a,'ATTRIBUTE'))

has_attribute_value(r,a,v)에서 r, a는 각각 RESOURCE, ATTRIBUTE 타입이며 v는 속성 a의 도메인 원소이다(Rule12).

Rule12: ∀r∀a∀v(∃i(has_attribute_value(i,r,a,v)) ⇒is_a(_,r,'ACCESS-NAME')∧is_a(_,a,'ACCESS-NAME') ∧ is_a(_,r,'RESOURCE') ∧ is_a(_,a,'ATTRIBUTE') ∧ instance_of(_,v,domain_of(a)))

has_relation(i,r,rel)에서 r과 rel은 각각 RESOURCE 및 RELATION 타입이다(Rule13).

Rule13: ∀r∀rel(∃i(has_relation(i,r,rel)) ⇒ is_a(_,r, 'ACCESS-NAME')∧is_a(_,rel,'ACCESS-NAME')∧is_a(_,r,'RESOURCE') ∧ is_a(_,rel,'RELATION'))

has_constraints(i,r,c)에서 r과 c는 access-name이다(Rule14).

Rule14: ∀r∀c(∃i(has_constraints(i,r,c)) ⇒ is_a(_,r, 'ACCESS-NAME')∧is_a(_,c, 'ACCESS-NAME'))

4.2 정보자원 구성에 대한 FOPL 표현

3.2절과 4.1절에서 정의된 atomic formula와 predicate 구성 제약조건 규칙은 3.1절에서 제안한 정보자원 요소들의 구성체계를 정의하기 위해 사용된다. 4.2절은 정보자원 요소들의 구성체계를 FOPL로 표현된 다음의 사실들과 규칙들을 생성하는 내용에 대하여 논한다.

IRD는 정보자원들로 구성된다. 따라서 다음의 규칙이 성립된다(Rule15).

Rule15: ∀ir(is_a(ir,'RESOURCE-INFO') ⇒ instance_of(ir, 'IRD'))

정보자원 구조와 내용을 나타내는 사실들(facts)을 3.1절의 그림.3과 atomic formula를 이용하여 표현하면 다음과 같다(F1-F11) :

Facts

F1: is_a('F1','RESOURCE-MANAGEMENT-INFO', 'RESOURCE-INFO')

F2: is_a('F2','RESOURCE-CREATION-INFO', 'RESOURCE-MANAGEMENT-INFO')

F3: is_a('F3','SECURITY-INFO', 'RESOURCE-MANAGEMENT-INFO')

F4: is_a('F4','CHANGE-INFO', 'RESOURCE-MANAGEMENT-INFO')

F5: is_a('F5','VERSION-CONTROL-INFO', 'RESOURCE-MANAGEMENT-INFO')

F6: is_a('F6','RESOURCE-REPRESENTATION-INFO', 'RESOURCE-INFO')

F7: is_a('F7','ROLE-INFO', 'RESOURCE-REPRESENTATION -INFO')

F8: is_a('F8','COMPOSITION-INFO', 'RESOURCE-REPRESENTATION-INFO')

F9: is_a('F9','ATTRIBUTE_INFO', 'RESOURCE-REPRESENTATION-INFO')

F10: is_a('F10','RELATIONSHIP-INFO', 'RESOURCE-REPRESENTATION-INFO')

F11: is_a('F11','CONSTRAINTS-INFO', 'RESOURCE-REPRESENTATION-INFO')

위의 F1에서 F11의 사실들은 정보자원의 구성을 나타내고 있다. 사실 F1과 F6으로부터 정보자원('RESOURCE-INFO')은 정보자원 관리요소('RESOURCE-MANAGEMENT-INFO')와 정보자원 표현요소('RESOURCE-REPRESENTATION-INFO')로 구성됨을 나타낸다. F2에서 F4는 자원관리정보('RESOURCE-MANAGEMENT-INFO')는 자원 생성정보(F2의 'RESOURCE-CREATION-INFO'), 보안정보(F3의 'SECURITY-INFO'), 자원변경정보(F4의 'CHANGE-INFO'), 버전제어정보(F5의 'VERSION-CONTROL-INFO') 등으로 구성됨을 의미한다. 한편, 자원표현 정보('RESOURCE-REPRESENTATION-INFO')는 다시 역할 정보(F7의 'ROLE-INFO'), 구성 정보(F8의 'COMPOSITION-INFO'), 속성 정보(F9의 'ATTRIBUTE-INFO'), 관계성 정보(F10의 'RELATIONSHIP-INFO'), 그리고 제약조건 정보(F11의 'CONSTRAINTS-INFO') 등으로 구성된다.

다음 제약조건들은 정보자원 관리요소를 구성하는 정보자원 생성 정보, 정보자원 변경 정보가 각각 갖추어야 할 속성을 나타낸다.

정보생성 정보는 속성으로 정보를 생성한 생성자('ADDED-BY')와 생성 시간('DATE-ADDED')을 갖는다(Rule16).

Rule16: ∀ir(is_a(_,ir, 'RESOURCE-CREATION-INFO') ⇒ ∃r∃u∃d(has_attribute(ir,r,u) ∧ is_a(_,u,'ADDED-BY')∧ is_a(_,d,'DATE_ADDED'))

정보변경 정보는 속성으로 정보를 변경한 변경자('USER')와 변경 시간('TIME'), 변경대상('SOURCE'), 변경내용('CONTENTS'), 그리고 변경이유('RATIONALE')를 갖는다(Rule17).

Rule17: ∀ir(is_a(_,ir,'CHANGE-INFO'))⇒ ∃u∃t(has_attribute(_,ir,u)∧is_a(_,u,'USER')∧has_attribute(_,ir,t)∧ has_attribute(_,r,s) ∧ is_a(_,a, 'SOURCE') ∧is_a (_,t,'TIME') ∧ has_attribute (_,r, c)∧ is_a(_,c, 'CONTENTS') ∧ has_attribute(_,ir,r) ∧is_a(_,r,'RATIONALE'))

역할 정보는 관련 activity와 연산 정보를 갖는다. 또한, activity정보와 연산정보를 갖는 정보는 역할 정보이다(Rule18).

Rule18: ∀ir(is_a(_,ir,'ROLE-INFO'))⇔ ∃r∃a∃role(resource_info_on(_,ir,r)∧has_role(_,r,a,role)∧has_operation(_,r,a,role,o))

연산 정보는 입력자원 또는 출력정보자원을 갖는다(Rule19).

Rule19: ∀r∀a∀role∀o(has_operation(_,r,a,role,o) ⇔ ∃input_r∃output_r(has_input(_,r,a,role, o,input_r) ∨ has_output(_,r,a,role,o,output_r) ∧is_a(_,input_r,'RESOURCE')∧ is_a(_,output_r,'RESOURCE'))

앞의 규칙들로부터 다음의 규칙들이 추론될 수 있다(Rule20-Rule34).

Rule20: ∀i(∃a∃b(a_part_of(i, a, b)∧is_a(_,a, 'RESOURCE')∧is_a(_,a,'RESOURCE')) ⇔ is_a(i,'COMPOSITION-INFO'))

Rule21: ∀i(∃r∃a∃role(has_role(i,r,a,role)) ⇔ is_a(i,'ROLE-INFO'))

Rule22: ∀i(∃r∃a∃role∃o(has_operation(i,r,a,role,o)) ⇔ is_a(i,'ROLE-INFO'))

Rule23: ∀i(∃r∃a(has_attribute(i,r,a)) ⇒ is_a (i,'ATTRIBUTE-INFO'))

Rule24: ∀i(∃r∃a∃v(has_attribute_value(i,r,a,v)) ⇒ is_a(i,'ATTRIBUTE-INFO'))

Rule25: ∀i(∃r∃rel(has_relation(i,r,,rel)) ⇔ is_a (i,'RELATIONSHIP-INFO'))

Rule26: ∀i(∃r∃c(has_constraints(i,r,c)) ⇔ is_a (i,'CONSTRAINTS-INFO'))

Rule27: ∀i(∃a∃b(a_part_of(i,a,b)∧is_a(_,a, 'RESOURCE') ∧is_a(_,a,'RESOURCE')) ⇔ resource_info_on(,i,a) ∧ resource_info_on(i,b))

Rule28: ∀i(∃r∃a∃role(has_role(i,r,a,role)) ⇒ resource_info_on(i,r))

Rule29: ∀i(∃r∃a∃role∃o(has_operation(i,r,a,role,o)) ⇒ resource_info_on(i,r))

Rule30: ∀i(∃r∃a(has_attribute(i,r,a)) ⇒ resource_info_on(i,r))

Rule31: ∀i(∃r∃a∃v(has_attribute_value(i,r,a,v)) ⇒ resource_info_on(i,r))

Rule32: ∀i(∃r∃rel(has_relation(i,r,,rel)) ⇒ resource_info_on(i,r))

Rule33: ∀i(∃r∃c(has_constraints(i,r,c)) ⇒ resource_info_on(i,r))

Rule34: ∀i(∃r(resource_info_on(i,r)) ⇔ (∃a∃b(a_part_of(i,a,b)∧is_a(_,a,'RESOURCE')∧ is_a(_,a,'RESOURCE'))) ∨(∃r∃a∃role(has_role(i,r,a,role))∨(∃r∃a

∃role∃o(has_operation(i,r,a,role,o))∨(∃r∃a(has_attribute(i,r,a))∨(∃r∃a∃v(has_attribute_value(i,r,a,v))∨(∃r∃rel(has_relation(i,r,,rel))∨(∃r∃c(has_constraints(i,r,c)))

4.3 상속 규칙(inheritance rules)

확장된 모델은 객체지향 모델에 기반을 두고 있기 때문에 클래스와 하위 클래스와의 관계 또는 클래스와 인스턴스와의 관계 측면(즉, is_a 측면)에서 비롯되는 상속성(inheritance)을 나타내는 상속 규칙(inheritance rule)들이 제시될 필요가 있다. 상속규칙들이 식별되어 Rule35에서 Rule42까지 FOPL로 다음과 같이 표현된다.

Rule35: ∀i(∃i(is_a(i,a,b)) ⇔ subclass_of(_,a,b) ∨ instance_of(_,a,b))

Rule36: ∀i∀b(∃i1∃i2∃a(is_a(i1,a,b) ∧ instance_of(i2,i,a))⇒instance_of(_,i,b))

Rule37: ∀a(∃i1∃i2∃b∃c1(is_a(i1,a,b) ∧ a_part_of(i2,c1,b)) ⇒ ∃c2(is_a(_,c2,c1)∧a_part_of(_,c2,a)))

Rule38: ∀a∀act∀role(∃i1∃i2∃b(is_a(i1,a,b) ∧ has_role(i2,b,act,role))⇒has_role(_,a,act,role))

Rule39: ∀a∀act∀role∀o(∃i1∃i2∃b(is_a(i1,a,b) ∧ has_operation(i2,b,act,role,o))⇒ has_operation(_,a,act,role,o))

Rule40: ∀a∀att(∃i1∃i2∃b(is_a(i1,a,b)∧has_attribute (i2,b,att)) ⇒has_attribute(_,a,att))

Rule41: ∀a∀rel(∃i1∃i2∃b(is_a(i1,a,b)∧has_relation (i2,b,rel)) ⇒ ∃rel1(has_relation(_,a,rel1)∨∃rel1(is_a(rel1, rel)∧has_relation(_,a,rel1)))

Rule42: ∀a∀c(∃i1∃i2∃b(is_a(i1,a,b)∧has_constraint(i2,b,c))⇒has_constraint(_,a,c))

위의 규칙들 중에서 Rule35는 is_a 측면 관계가 subclass_of 측면 관계나 instance_of 측면 관계임을 의미한다. Rule36은 어느 한 클래스의 인스턴스는 그 클래스의 상위 클래스의 인스턴스가 됨을 의미한다. Rule35부터 Rule41들은 한 클래스의 하위 클래스나 인스턴스는 그 클래스로부터 구성요소, 역할, 연산, 속성, 관계, 제약조건을 상속받음을 의미한다.

5. FOPL 규칙 변환과 관리

IRD 확장 모델을 기반으로 FOPL 형태로 표현된 predicate들과 규칙들은 각각 prolog predicate들과 규칙들로 구현 가능하다. 정보자원의 사용자는 자원에 대한 각 종 정보를 얻기 위해 질의를 이용한다. 여러 사용자층을 대상으로 질의 표현 환경을 지원하기 위하여 질의 형식과 표현은 단순성을 갖추어야 하며, 다양한 형태의 질의를 구사할 수 있도록 융통적이어야 한다. 또한, FOPL은 기본적으로 제약조건의 표현과 atomic formula와 규칙을 이용하여 추론을 잘 수행하는 구현 언어를 필요로 한다. prolog는 이러한 요구조건에 적합한 구현언어이다. FOPL을 prolog로 구현하는 방법은 논리(logic)에 근거한 이론들에 의해 비교적 수월하게 이루어진다.

이를 위해 5.1 절에서는 FOPL로 표현된 IRD 확장 모델의 predicate들과 규칙들을 prolog 규칙들로 변환하는 메커니즘을 제시하고 정보자원 관리를 위해 필요로 되는 prolog predicate들을 제안하였다. assert된 predicate들과 규칙들이 주어질 때 prolog는 backward-chaining 메커니즘으로 추론을 행한다. prolog에서 질의는 assert된 predicate의 파라메터로서 대문자 변수를 이용하거나, p :- q 형태의 규칙에서 결론에 해당되는 p를 이용하여 행하거나 혹은 이들의 and/or 조합에 의해 다양한 형태의 질의를 행할 수 있다. prolog는 또한 동적으로 새로운 predicate들과 규칙들을 첨가할 수 있으므로 앞의 요구사항들을 수용할 수 있는 적합한 언어이다.

한편, 5.2절은 정보자원 관리를 위한 predicate들 중에서 정보자원 생성을 위한 predicate들을

제안하고 스키마 관리 방법론을 제안한다. 5.3절은 사용자 질의 구성 예를 보이면서 predicate을 이용한 질의 구성 방법을 간략하게 언급한다.

5.1 FOPL의 변환 방법론

FOPL의 규칙들은 prolog 형태로 변환된다. prolog 문장은 기본적으로 가정과 결론의 형태, 즉, 의미상 if p then q 의 기본형태를 취하며 구문 상 q :- p 형태로 :- 앞에는 결론 q가 뒤에는 가정 p가 각각 위치한다. 앞에서 FOPL로 제안된 규칙들은 다음의 3가지 형태를 취한다 (여기에서 pi와 qi는 predicate이며 기호 '∨∧'은 'and 혹은 or'을 의미한다.).

형태.1 $p1 \wedge p2 \wedge p3 \wedge ... \wedge pn \Rightarrow q1 \wedge q2 \wedge ... \wedge qm$
형태.2 $p1 \vee p2 \vee p3 \vee ... \vee pn \Rightarrow q1 \wedge q2 \wedge ... \wedge qm$
형태.3 $p1 \vee \wedge p2 \vee \wedge p3 \vee \wedge ... \vee \wedge pn \Rightarrow q1 \vee \wedge q2 \vee \wedge ... \vee \wedge qm$

형태.1의 규칙은

$(p1 \wedge p2 \wedge p3 \wedge .. \wedge pn \Rightarrow q1 \wedge q2 \wedge ... \wedge qm) \equiv$
$(p1 \wedge p2 \wedge p3 \wedge ... \wedge pn \Rightarrow q1) \wedge$
$(p1 \wedge p2 \wedge p3 \wedge ... \wedge pn \Rightarrow q2) \wedge$
$(p1 \wedge p2 \wedge p3 \wedge ... \wedge pn \Rightarrow q3) \wedge$
.....$\wedge$
$(p1 \wedge p2 \wedge p3 \wedge ... \wedge pn \Rightarrow qm)$

이므로 뒤에서 설명될 변환.1 형태의 prolog로 변환된다.

형태.2의 규칙은

$(p1 \vee p2 \vee p3 \vee ... \vee pn \Rightarrow q1 \wedge q2 \wedge ... \wedge qm) \equiv$
$\neg(p1 \vee p2 \vee p3 \vee ... \vee pn) \vee (q1 \wedge q2 v ... \wedge qm) \equiv$
$(\neg p1 \wedge \neg p2 \wedge \neg p3 \wedge ... \wedge \neg pn)$ v $(q1 \wedge q2 ... \wedge qm) \equiv$
$((\neg p1 \wedge \neg p2 \wedge \neg p3 \wedge ... \wedge \neg pn) \vee q1) \wedge$
$((\neg p1 \wedge \neg p2 \wedge \neg p3 \wedge ... \wedge \neg pn) \wedge q2) \wedge$
.....$\wedge$
$((\neg p1 \wedge \neg p2 \wedge \neg p3 \wedge ... \wedge \neg pn) \vee qm) \equiv$
$(\neg p1 \wedge q1) \wedge (\neg p2 \vee q1) \wedge (\neg p3 \vee q1) \wedge ... \wedge$
$(\neg pn \vee q1) \wedge$
$(\neg p1 \vee q2) \wedge (\neg p2 \vee q2) \wedge (\neg p3 \vee q2) \wedge ... \wedge$
$(\neg pn \vee q2) \wedge$
.....$\wedge$
$(\neg p1 \vee qm) \wedge (\neg p2 \vee qm) \wedge (\neg p3 \vee qm) \wedge ... \wedge \neg$
$(pn \vee qm) \equiv$
$(p1 \Rightarrow q1) \wedge (p2 \Rightarrow q1) \wedge (p3 \Rightarrow q1) \wedge ... \wedge (pn \Rightarrow q1) \wedge$
$(p1 \Rightarrow q2) \wedge (p2 \Rightarrow q2) \wedge (p3 \Rightarrow q2) \wedge ... \wedge (pn \Rightarrow q2) \wedge$
.....$\wedge$
$(p1 \Rightarrow qm) \wedge (p2 \Rightarrow qm) \wedge (p3 \Rightarrow qm) \wedge ... \wedge (pn \Rightarrow qm)$

이므로 변환.2와 같이 prolog 형태로 변환된다.

한편, 형태.3은 결론부가 and와 or로 연결된 형태로서 prolog 구문 형태로의 직접적인 변환은 불가능하지만 제약조건 형태로

$(p1 \vee \wedge p2 \vee \wedge p3 \vee \wedge ... \vee \wedge pn \Rightarrow q1 \vee \wedge q2 \vee \wedge ... \vee \wedge qm)$
$\equiv \neg(p1 \vee \wedge p2 \vee \wedge p3 \vee \wedge ... \vee \wedge pn) \vee (q1 \vee \wedge q2 \vee \wedge ... \vee \wedge qm) \equiv$
$(\neg p1 \wedge \vee \neg p2 \wedge \vee \neg p3 \wedge \vee ... \wedge \vee \neg pn) \vee (q1 \wedge \vee q2 \wedge \vee ... \wedge \vee qm)$

와 같이 유도되어 변환.3의 형태로 변환된다.

변환.1(형태.1의 변환)

```
q1 :- p1,p2,p3,...,pn.
q2 :- p1,p2,p3,...,pn.
q3 :- p1,p2,p3,...,pn.
.....
qm :- p1,p2,p3,...,pn.
```

또는,

```
constraint(i,c) :- not(p1,p2,p3,...,pn) or ( q1,q2,...,qm).
```

변환.2(형태.2의 변환)

```
q1:- p1.
```

q1:- p2.
.....
q1:- pn.
q2:- p1.
q2:- p2.
.....
q2:- pn.
.....
qm:- p1.
qm:- p2.
.....
qm:- pn.

또는,

constraint(i,c) : - not(p1), not(p2), not(p3),..., not(pn) or (q1.q2,..,qm).

변환.3(형태.3의 변환)

constraint(i,c) : - (not(p1) and/or not(p3) and/or ...and/or not(pn)) or (q1 and/or q2 and/or...and/or qm).

5.2 정보자원 관리지원을 위한 기본 predicate들

정보자원 관리는 크게 스키마 관리, 인스턴스 관리, 무결성 관리를 포함한 규칙 관리 등으로 구분된다. 이들에 대한 관리는 기본적으로 새로운 정보자원의 생성(첨가), 정보자원 내용의 변경, 정보자원 내용의 삭제, 정보자원 내용의 검색 및 추론을 포함하며 때로는 무결성 검사와 질의 처리를 위해 규칙들을 이용한 추론 실행이 요구된다. 정보자원 내용 변경은 atomic formula 사실 단위로 행해지며 그 변경하고자 하는 원래의 정보자원을 모든 기본 atomic formula와 직접적으로 결부된 사실들을 모두 추론하여 이들 각각에 대해 적절한 처리를 규칙에 의해 자동 혹은 수동으로 처리해야 하며 이 처리 규칙은 응용 도메인에 따라 다양하다. 각종 변경을 통하여 변경이전의 정보와 새로운 정보간에 버전관계를 형성한다. 그러나 이 버전관계와 관리는 응용에 따라 그 해결방법이 다양하고 본 논문의 범위를 초월하므로 다루지 않도록 한다.

5.2.1 정보자원 생성 predicates

응용 분야에 따라 정보자원 생성에 위한 정보는 여러 가지 있을 수 있지만 기본적으로 정보자원 표현요소 및 그 정보자원 관리요소 등 두 가지 유형의 정보자원 생성이 요구된다. 정보자원 관리를 위해서는 정보자원이 생성될 때 그 정보에 대한 식별자의 생성도 요구된다. 또한, 이 식별자외에 그 정보자원을 생성한 사용자와 생성시기에 관한 자료 사전적 정보 생성도 수반된다. 이를 위한 기본 predicate들과 그 기능들은 다음과 같다.

generate_Rid(Rid) : 정보 식별자를 생성하는 기능을 갖는다. 정보 식별자를 생성하여 변수 Rid로 할당한다.

get_Uid(Uid) : 시스템을 사용하고 있는 사용자의 식별자를 변수 Uid로 할당한다.

get_Time(Time) : 현재의 날짜와 시간을 구하여 변수 Time에 할당한다.

assert(p(a,b,...,c)) : 여기서 p(a,b,...,c)는 사실을 나타내는 formula이며 IRD의 p predicate들의 집합에 p(a,b,...,c)를 삽입한다.

자원 자체 정보생성을 위한 predicate들의 종류와 그 기능들은 다음과 같다.

create_is_a(a,b) : 정보 관리를 위한 정보자원 식별자 i를 구하여 is_a(i,a,b)

```
create_is_a(A,B) :-
        \+(is_a(_,A,B)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(is_a(i,A,B)),
        assert(is_a(j,i,"RESOURCE_CREATION_INFO")),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),

        assert(has_attribute_value(j,"DATE_ADDED",Time)).
create_instance_of(A,B) :-
        \+(instance_of(_,A,B)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(instance_of(i,A,B)),
        inherit_assert_instance_info(Uid,Time,A,B),
        assert(is_a(j,i,"RESOURCE_CREATION_INFO")),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_a_part_of(A,B) :-
        \+(a_part_of(_,A,B)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(a_part_of(i,A,B)),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_has_role(R,A,ROLE) :-
        \+(has_role(_,R,A,ROLE)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(a_part_of(i,R,A,ROLE)),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_has_operation(R,A,ROLE,O) :-
        \+(has_operation(_,RA,ROLE,O)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(has_operation(i,R,A,ROLE,O)),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_has_attribute(R,A) :-
        \+(has_attribute(_,R,A)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(has_attribute(i,R,A)),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_has_attribute_value(R,A,V) :-
        \+(has_attribute_value(_,RA,V)),
        generate_Rid(i), generate_Rid(j),
        get_Uid(Uid), get_Time(Time),
        assert(has_attribute_value(i,R,A,V)),
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)).

create_has_relation(R,REL) :-
        \+(has_relation(_,R,REL)),
        generate_Rid(i), generate_Rid(k),
        get_Uid(Uid), get_Time(Time),
        assert(has_relation(i,R,REL)),
        create_resources_info_on(R,REL),
        assert(has_attribute_value(k,"ADDED_BY",Uid)),
        assert(has_attribute_value(k,"DATE_ADDED",Time)).

create_resources_info_on(R,REL) :- \+(has_relation(_,R,REL)),
        get_Uid(Uid), get_Time(Time),
        generate_Rid(j),
        assert(resources_info_on(j,R,REL)))
        assert(has_attribute_value(j,"ADDED_BY",Uid)),
        assert(has_attribute_value(j,"DATE_ADDED",Time)),!.

create_resources_info_on(_,_).
```

그림 5: 정보자원 생성을 위한 함수의 구현 예

를 삽입하고 관리 정보를 삽입한다.

create_instance_of(a,b) : 정보 관리를 위한 정보자원 식별자 i를 구하여 instance_of(i,a,b)를 삽입하고 관리 정보를 삽입한다.

create_a_part_of(a,b) : 정보 관리를 위한 정보자원 식별자 i를 구하여 a_part_of (i,a,b)를 삽입하고 관리 정보를 삽입한다.

create_has_role(r,a,role) : 정보 관리를 위한 정보자원 식별자 i를 구하여 has_role(i,r,a,role)를 삽입하고 관리 정보를 삽입한다.

create_has_operation(r,a,role,o) : 정보 관리를 위한 정보자원 식별자 i를 구하여 has_operation(i,r,a,role,o)을 삽입하고 관리 정보를 삽입한다.

create_has_attribute(r,a) : 정보 관리를 위한 정보자원 식별자 i를 구하여 has_attribute(i,r,a)를 삽입하고 관리 정보를 삽입한다.

create_has_attribute_value(r,a,v) : 정보 관리를 위한 정보자원 식별자 i를 구하여 has_attribute_value (i,r,a,v)를 삽입하고 관리 정보를 삽입한다.

create_has_relation(r,rel) : 정보 관리를 위한 정보자원 식별자 i를 구하여 has_relation(i,r,a,rel)과 resources-info-on(j,r,rel)을 삽입하고 관리 정보를 삽입한다.

위의 정보자원 생성 기능들에서 정보자원 관리요소의 생성은 자료 사전적 정보로서 응용영역과 시스템에 의존한다. 정보자원 생성을 위한 한 예가 그림 5에서 제시된다.

5.2.2 스키마 관리

스키마 관리는 클래스들 사이의 관련성에 의해 형성된 계층구조의 변경과 클래스 자체가 소유한 속성, 역할, 연산, activity, 제약조건의 삽입, 삭제, 변경을 지원해야 한다. 스키마 변경 종류들과 그 관리를 위한 주요 처리들을 세부적으로 구분하면 다음과 같다.

- 클래스 명칭 변경
  클래스의 명칭 변경은 원래의 명칭과 관련된 모든 atomic formula들을 추론하여 이들 각각에 대해 새로운 명칭으로 대체함으로써 이루어진다.
- 클래스들의 계층구조 변경
  클래스들의 계층 구조 변경은 is_a 계층구조와 a_part_of 계층구조 그리고 이들 관계를 모두 고려한 계층구조 세 가지 측면에서 고려되어야 한다. 클래스의 삽입과 삭제에 의해 이루어진다. 클래스의 삽입에서 삽입되는 클래스가 최상위 클래스의 상위 클래스로서 기존의 최상위 클래스와 is_a 관계를 삽입하여 그 계층구조를 변경할 수 있으나 상속(inheritance)에 의해 발생할 수 있는 하위 클래스들 사이 그리고 소속 인스턴스들의 속성, 역할, 부분 클래스, 연산, 관계, 제약조건 들에 대한 충돌과 불일치성을 검사하여 처리해야 한다(물론 이 충돌해결은 규칙에 의해 자동으로 해결할 수도 있고 수동으로 해결할 수도 있다). 중간 클래스로서 삽입될 경우에는 예정된 하위 클래스 및 상위 클래스들과 is_a 관계를 삽입함과 동시에 최상위 클래스들로부터 최하위 클래스 및 소속 인스턴스에 이르기까지 상속으로 인해 발생 가능한 속성, 역할, 부분 클래스, 연산, 관계, 제약조건 들에 대한 충돌과 불일치성을 검사하여 처리해야 한다. 최하위 클래스의 하위 클

래스로 삽입될 경우 최상위 또는 중간 클래스의 삽입에서 발생되는 유형의 충돌과 불일치성의 검사는 물론 최하위 클래스에 소속되었던 인스턴스들의 전부 또는 일부를 소속 변경할 것인가에 대한 결정이 이루어져야 한다.

a_part_of 관계에 의한 부분 클래스의 삽입은 앞의 is_a 관계 삽입에 의한 처리와 유사하게 a_part_of 관계의 삽입으로 상위 또는 하위 클래스와의 구성관계를 반영할 수 있으나 간과해서 안될 것은 그 클래스를 구성요소로 갖고 있는 클래스들의 상위 클래스 및 하위 클래스들, 그 클래스를 구성요소로 갖는 클래스들, 그리고 그 클래스를 구성하는 부분 클래스들 간의 속성, 역할, 부분 클래스, 연산, 관계, 제약조건들에 대한 충돌과 불일치성을 검사하여 처리해야 한다.

계층구조에서 클래스의 삭제는 상위 및 하위 클래스들간의 관련성 제거, 클래스 자체가 소유한 속성, 역할, 연산, 관계, 그리고 그 클래스와 관련된 제약 조건과 규칙들의 처리문제, 삭제되었을 경우의 제약조건에 의한 무결성 검사 문제의 해결과 소속 인스턴스들의 처리 문제 등이 결정되어야 한다. 삭제되는 클래스를 입출력으로 이용하는 연산의 식별과 처리도 고려되어야 한다. 또한, 삭제되는 클래스와 관련된 제약조건들 및 규칙들의 식별과 변경여부도 처리되어야 한다.

- 클래스 자체의 변경

  클래스의 변경은 속성, 역할, 연산, 관계, 제약조건 들의 변경, 삽입, 삭제에 의해 유발된다. 속성, 역할, 관계, 제약조건은 모두 관련 클래스 혹은 인스턴스들에 의해 식별된다. 각각에 대한 세부처리 작업은 다음과 같다.

- 속성 삽입

  해당 클래스로의 속성 삽입은 속성 명칭, 속성 타입 또는 속성 값 등의 정보 결정 및 삽입 처리와 부수적으로 행해져야하는 관련된 제약조건 또는 규칙의 삽입 또는 변경 처리, 그리고 해당 클래스의 상위 및 하위 클래스들간에 상속에 의해 유발되는 발생되는 중복 정의 및 불일치성 문제 등을 해결해야 한다. 또한, 소속 인스턴스들 전부 또는 일부에 대해 해당 클래스에 삽입되는 속성의 값의 삽입여부가 결정되어야 하며 새로운 속성과 연관된 연산, 제약조건 등의 첨가 여부를 결정해야 한다.

- 속성 명칭 변경

  속성 명칭의 변경은 변경 후의 속성 명칭과 상위 또는 하위 클래스간의 명칭 충돌 검색과 충돌 해결 처리, 원래의 이 속성을 입력자원 또는 출력자원으로 이용하는 연산의 식별과 처리, 속성에 부여된 제약조건들 및 규칙들의 추론검색과 이들 제약조건과 규칙이 포함하는 변경이전의 속성 명칭을 새로운 속성 명칭으로 변경, 그리고 소속 인스턴스의 속성 명칭 변경 등의 처리 작업을 요구한다.

- 속성 타입 변경

  속성의 타입 변경은 속성이 atomic 클래스인 경우와 복합(composite) 클래스 경우의 두 측면에서 고려되어야 한다. atomic 클래스인 속성 타입의 변경은 하위 클래스들에 대하여 동일 속성의 기존 타입과 해당 값에 대한 검사를 필요로 하며 타입변경으로 인한 값의 변경이 요구된다. 또한 그 속성을 갖는 클래스들 각각에 소속한 인스턴스들이 추론되어 변경된 속성타입에 일치하는 값으로의 변경도 요구된다.

  속성이 복합 클래스인 경우의 타입 변경은 스키마의 변경과 동일한 원인(즉, 속성 자체 혹은 하위레벨들을 구성하는 클래스, 속성, 역할, 연산, 관계성, 제약조건 변경

등)이므로 스키마 변경 종류 분석과 관리는 스키마 변경 관리에 대한 지금까지 논의된 내용들과 앞으로 논의될 내용들이 모두 고려되어야 한다.

- 속성 값 변경
  속성 값의 변경은 변경된 속성 값이 속성 타입과의 일치성 검사와 처리, 그리고 그 속성과 관련된 연산 및 제약조건 검사와 처리가 요구된다.
- 속성 삭제
  속성이 단일 속성인가 혹은 복합 속성인가에 따라 관리 방식이 구분된다. 단일 속성 삭제는 그 속성을 갖는 클래스 및 하위 클래스들에 소속된 인스턴스들의 속성 값 삭제 처리, 관련된 연산 및 제약조건들의 검색과 변경, 그리고 추론 규칙들의 검색과 변경이 요구된다. 복합 속성의 경우 그 자체가 클래스로 간주되므로 그 속성의 삭제는 클래스의 삭제와 동일하게 처리되어야 한다.
- 속성 값 삭제
  속성 값 자체에 의존하는 연산, 제약조건, 규칙들의 식별과 처리가 요구되며 속성 값의 상속과 관련하여 이 속성과 관련된 클래스의 하위 클래스와 관련 인스턴스의 해당 속성 값의 삭제 여부를 결정해야 한다.
- 역할 삽입
  해당 클래스에 역할을 삽입하고 상속 여부를 결정하고 처리한다.
- 역할 명칭 변경
  역할 명칭의 변경은 역할 수행을 위한 연산들과의 연관 관계에서 역할 이름의 변경을 요구한다. 또한 원래 명칭의 역할에 부여된 제약조건과 규칙들의 식별과 변경도 요구된다.
- 역할 삭제
  역할의 삭제는 관련된 연산들의 삭제여부의 결정을 필요로 한다. 또한 관련된 제약조건 및 규칙들의 식별과 처리여부도 결정되어야 한다.
- 연산 삽입
  연산의 삽입은 그 연산을 수행하는 객체와 그 객체의 역할과의 관계성을 IRD에 삽입함을 의미한다. 연산의 삽입 시 연산과 관련된 제약조건 및 규칙들의 식별과 각각에 대한 검사가 요구될 수도 있다.
- 연산 명칭 변경
  연산의 명칭 변경에 IRD 내에서 원래의 연산 명칭과 관련된 객체와 역할을 찾아서 새로운 연산 명칭과 관련시켜야 한다. 또한, 원래의 연산 명칭을 이용하던 제약조건과 규칙들을 식별하여 변경여부를 결정하고 처리해야 한다.
- 연산의 입출력 자원 변경
  연산의 입출력 자원 변경은 그 연산과 관련된 입출력 자원의 명칭 변경에 의해 수행된다. 따라서 그 연산 및 그 연산의 입출력과 관련된 제약조건과 규칙들을 식별하여 변경여부를 결정하고 처리해야 한다.
- 관계 삽입
  관계의 삽입은 관계 명칭과 그 관계에 참여하는 자원들의 결정과 이들을 IRD로 삽입하는 과정이다.
  has_relation(i,r,'r_name')
  resources_on_relation(j,r,'r_name')
  (여기에서 r은 자원 명칭을 나타내며 'r_name'은 관계성 명칭을 각각 나타낸다. 관계에서 cardinality는 제약조건에 의해 해결될 수 있다)
- 관계 명칭 변경
  관계 명칭 변경은 resources_on_relation (i,r,'r_name') predicate과 has_relation(j,r,'r_name') predicate들을 찾아서 원래의 관계 명칭 'r_name'을 다른 명칭으로 변경한다. 변경되는 관계를 포함하는 제약조건과 규칙들을 식별하여 변경여부를 결정하고 처

리해야 한다.

- 관계 대상 변경
  관계 명칭 변경은 resources_on_relation (i,r,'r_name') predicate에서 관계의 대상이 되는 자원 r을 다른 자원으로 변경하는 것을 의미한다. 관계 대상 변경은 관계성에 새로운 자원 r의 참여, 관계성에 참여하는 기존의 자원 명칭 변경, 관계성에 참여하는 자원의 삭제 등으로 구분된다. 관계성에 새로운 자원 r을 참여시키는 경우에는 resources_on_relation(i,r','r_name') predicate를 삽입함으로써 해결된다. 관계성에 참여하는 기존의 자원 명칭 r의 변경은 대응하는 resources_on_relation(i,r,'r_name') predicate와 has_relation(j,r,'r_name') predicate들을 찾아서 원래의 관계 대상인 자원 r을 다른 대상으로 변경하고 관련된 제약조건과 규칙들을 식별하여 변경여부를 결정하고 처리해야 한다. 관계성에서 참여하고 있는 자원 r을 삭제할 경우에는 resources_on_relation(i,r,'r_name') predicate와 has_relation (j,r,'r_name') predicate들을 찾아서 삭제하고 관련된 제약조건과 규칙들을 식별하여 변경여부를 결정하고 처리해야 한다.
- 제약조건 또는 추론 규칙 변경
  위의 스키마 변경 처리는 부분적으로는 추론에 의해 자동으로 처리되나(예로서 변경과 관련된 객체들의 추론은 규칙에 의해 식별 가능하다) 부분적으로는 사용자에 의한 판단에 의해 처리되어야 한다. 사용자의 판단에 의한 처리가 정형화될 경우 새로운 규칙의 생성과 삽입에 의해 자동화될 수 있다. 그러나, 시스템이 지원할 수 있는 최대의 지원책은 변경에 따른 규칙의 적용과 제약조건 검사 및 변경과 관련된 IRD 대상들과 종류들을 추론하여 사용자에게 제공하고 일부는 사용자 판단에 의해 처리하도록 유도함으로써 IRD의 무결성을 최대한으로 유지시키고 자동관리를 위한 새로운 규칙과 제약조건의 변경과 확장을 가능하게 하는 환경을 제공해야 한다.

5.3 기본적인 사용자 질의

대부분의 연구들에서 사용자 질의의 작성은 비절차적 데이터 언어인 SQL을 이용하고 있으며 질의작성 대부분을 데이터 관리자에 의한 정의에 의존한다[4,5]. 그러나, FOPL과 이 종류의 prolog 언어는 도출연역에 의한 추론을 기본으로 제공하기 때문에 질의 작성이 매우 용이하다. 그 예로서 '속성-타입에 속하는 IRD 요소들을 모두 검색하라'에 대한 FOPL 및 prolog 질의는 다음과 같이 표현할 수 있다.

Q1 : is_a(_, E,'ATTRIBUTE-TYPE').

Prolog의 질의에서 첫자가 대문자로 표기된 스트링은 추론 결과가 바인딩되는 변수를 의미하며, 밑줄 '_'은 어떤 값이든 무관함을 나타낸다. 위의 간단한 질의에서 추론 결과는 is_a(i,a,b) 형태의 사실들(추론될 사실들도 포함)에서 b가 'ATTRIBUTE-TYPE'인 모든 a 값들이 변수 E로 할당되어 출력된다.

질의는 기본 측면을 이용하여 구성된 기본 predicate들, 관계/논리연산, 지정문 들의 and/or/not 및 조건문, 입출력문들과 prolog문으로 작성된 반복문들로 구성될 수 있다. 그러나, 기본적인 모든 단순한 질의들은 대부분 기본 predicate들과 논리 연산의 조합으로 구성된다. 예를 들어 '무게(weight)가 30kg 이상이면서 바퀴(wheel)를 구성요소로 갖는 자원을 모두 찾아라'와 같은 질의는

Q2: has_attribute_value(_,R,'weight',W),W>=30, a_part_of(_,'wheel',R)

와 같이 기본 predicate들과 비교연산의 조합으로 매우 단순하게 표현될 수 있다(위의 질의에

서 콤머 ','는 and 연산을 나타낸다). 위의 질의는 속성 'weight'의 단위가 'kg'을 기본값으로 취할 경우 표현할 수 있으며 필요에 따라 'weight-unit'이라는 속성을 has_attribute_value(_,R,'weight-unit', 'kg') 형태로 첨가하여 무게 단위를 비교하면서 질의를 행할 수도 있다.

6. 결 론

본 논문은 FIPS IRDS가 안고 있는 문제점을 논하고, 이에 대한 해결을 위해 IRD 확장 모델을 제안하여 이 모델 요소들과 이 요소들 사이에 존재하는 제약조건과 규칙을 식별하여 FOPL로 표현하였으며, prolog를 이용한 FOPL 기반의 IRDS 구현에 관하여 논하였다. 서론에서도 언급한 바와 같이 본 논문에서 제안한 IRD의 확장 모델과 정보자원 표현을 위한 FOPL의 도입, 그리고 prolog로의 IRDS 구현은 FIPS IRDS가 갖는 문제점의 해결과 아울러 정보자원 표현의 자연성을 제공하고, 정보자원을 필요로 하는 사용자에게 질의 작성을 수월하게 하고 진보된 추론 기능을 제공하는 이점을 제공한다.

추론과정에 있어 prolog의 exhaustive 탐색으로 인한 성능저하(본 논문에서는 성능에 대하여는 고려하지 않았다)는 기존에 제안된 관계 DB를 이용한 IRDS 구현과 비교해 볼 때 앞의 장점들과 trade-off 관계를 갖는다. 그러나, 이 문제는 기존의 DB와 prolog의 인터페이싱을 통하여 어느 정도 해결될 수 있으리라 본다. 예를 들어 사실들(fact)에 해당하는 정보자원들을 DB내로 표현하고 이들 사이에 존재하는 제약조건과 규칙들을 prolog로 표현함으로써 데이터 탐색에 요구되는 시간을 단축하고 제약조건과 추론 기능을 강화할 수 있다(이에 대한 필요성과 그 구현에 대한 내용들이 연역 데이터 베이스를 중심으로 이미 많은 연구들이 행해져 왔다).

한편, 본 논문에서 제안한 질의를 위한 기본 predicate들은 사실상 IRD 확장 모델을 이루는 측면-타입의 구성요소들이므로 is_a(_,Q,'ASPECT-TYPE') 질의로 그 종류들을 변수 Q에 바인딩되도록 함으로써 검색할 수 있다. 그러나, 어떤 predicate에 대한 정확한 지식이 없는 사용자가 질의를 위해 또는 재사용을 위해 그 predicate를 정확히 사용할 수 있도록 위해서는 그 기능에 대한 지식을 사용자에게 부여하고, 그 predicate에 이용되는 파라메터의 순서와 타입을 제공해야 하기 때문에 이를 지원하기 위한 지식표현 방법이 더욱 연구되어야 한다.

〈References〉

[1] Abandham, M., *et. al.*, "A View of the IRDS Reference Model", *Database Programming & Design*, 1990, 3, pp41-53

[2] Allen, F.W., Loomis, M.E.S., Mannini, M.V., "The Integrated Dictionary/Directory System", *Computing Surveys*, Vol.14, No.2, 1982, pp27-67

[3] Kim, C.H., "REAPS: A Knowlege Base System for Sopporting REA Modelling", *Proc. of 2nd '93 korea/japan Expert system Joint Conference*, 1993, pp633-650

[4] Dolk, D.R., Kirsch, R.A., "A Relational Information Resource Dictionary System", *Communication of the ACM*, Vol.30, No.1, 1987, pp48-61

[5] Dolk, D. R., "Model Management and Structured Modelling: The Role of an Information Resource Dictionary System", *Communication of the ACM*, Vol.31, No.6, 1988, pp704-718

[6] Elaine Rich, *Artificial Intelligence*, McGRAW-Hill Book Company, 1983

[7] Fadal, F.G., *"Resource Ontology"*, in The Toronto Virtual Enterprise Model, 1994, pp45-104

[8] Frost, R.A., *Introduction to Knowledge Base Systems*, Collins Professional and Technical Books, 1986

[9] Guy Eddon and Henry Eddon, *Inside Distributed COM*, Microsoft press., 1998.

[10] Hazzah, A., "Data Dictionaries: Paths to a Standard", *Database Programming & Design*, 1989, 8, pp26-32

[11] Hsu, C., Bouziane, M., Yee, L., "Information Resources Management in Heterogeneous, Distributed Environments: A Metadatabase Approach", *IEEE Transactions on Software Engineering*, Vol.17, No.6, 1991, pp604-625

[12] ISO/IEC SC21/WG3 N1095, *Information Resource Dictionary System(IRDS) Services Interface*, ISO/IEC_JTC1/SC21, Information Retrieval, Transfer and Management for OSI, WG3 DATABASE, Project 1.21.6.2

[13] ISO/JTC 1/SC 21/WG3 N711, *Information Systems - Open Systems - Remote Database Access Tutorial*, ISO/IEC_JTC1/SC21, Information Retrieval, Transfer and Management for OSI, WG3 DATABASE, Project 1.21.31

[14] Moriarty, T., "Are You Ready for a Repository ?", *Database Programming & Design*, 1990, 3, pp61-71

[15] Oliver, H., A *Comparison of ISO-IRDS and PCTE*, Hewlett-Packard Lab., Bristol, 1992

[16] Parabandham, M., Selfridge, W. J., Mann, D.D., "The Role of the IRDS", *Database Programming & Design*, 1990, 6, pp41-48

[17] Reaz Hoque, *CORBA3*, IDC Books Worldwide, Inc., 1998.

[18] Tomlinson, C., Lavender, G., "The Carnot Extensible Services Switch(ESS)-Support for Service Execution", MCC *Technical Report Number Carnot-129-92*

[19] 김영실, 조동영, 백두권, 황종선, "IRDS 에서 버전 개변 사용 및 그 제어방법", _

[20] 김창화, *EA 모델에 의한 지식 표현 모델링*, 고려대학교 대학원, 박사학위논문, 1989.

[21] 조명섭, 김명준, "IRDS", *정보통신기술*, 제 2 권, 제 3 호, 1988, pp30-34