creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

[UCI]I804:24011-200000234431

2007년 8월

박사학위논문

의미적 정보검색을 위한 온톨로지 자동 구축 및 효율적 질의 처리 방법

조 선 대 학 교 대 학 원

전 자 계 산 학 과

공 현 장

의미적 정보검색을 위한 온톨로지 자동 구축 및 효율적 질의 처리 방법

The Method of Automatic Ontology Building and Efficient Query Processing for the Semantic Information Retrieval

2007년 8월 일

조 선 대 학 교 대 학 원

전 자 계 산 학 과

공 현 장

의미적 정보검색을 위한 온톨로지 자동 구축 및 효율적 질의 처리 방법

지도교수 김 판 구

이 논문을 이학박사학위 논문으로 제출함.

2007년 4월

조 선 대 학 교 대 학 원

전 자 계 산 학 과

공 현 장

공현장의 이학박사 학위논문을 인준함

위원장　　조선대학교 교수 _____印

위　원　　조선대학교 교수 _____印

위　원　　전남대학교 교수 _____印

위　원　　충남대학교 교수 _____印

위　원　　조선대학교 교수 _____印

2007년　6월

조선대학교 대학원

목 차

제 1 장 서 론 ····· 1

제 1 절 연구 배경 및 필요성 ····· 1
제 2 절 연구 내용 및 범위 ····· 2

제 2 장 관련연구 ····· 3

제 1 절 온톨로지와 시맨틱 웹 ····· 3
1. 온톨로지(Ontology) ····· 3
2. 시맨틱 웹 ····· 5

제 2 절 국내외 온톨로지 저작 도구 ····· 10
1. 기존 온톨로지 저작 도구 비교·분석 ····· 10

제 3 장 효율적인 온톨로지 처리 ····· 14

제 1 절 온톨로지 기반 의미적 정보검색 ····· 14
제 2 절 온톨로지 구축 ····· 17
제 3 절 온톨로지 파싱 ····· 25
제 4 절 온톨로지 질의 처리 ····· 37

제 4 장 구현 및 평가 ····· 48

제 5 장 결론 및 제언 ····· 57

참고문헌 ····· 59

부록 1. 추론 규칙 및 규칙에 대한 파싱 모듈 설계도 ····· 62
1. 추론 규칙 ····· 62
2. 추론 규칙에 대한 파싱 모듈 설계도 ····· 64

표 목 차

[표 1] OTK 방법론 ····· 4
[표 2] Ontology Development 101 방법론 ····· 4
[표 3] 기존 웹 기술과 시맨틱 웹 기술 특성 비교 ····· 5
[표 4] RDF 표현 ····· 8
[표 5] RDF 스키마 표현 ····· 8
[표 6] RDF 스키마를 사용한 Title 정의 ····· 9
[표 7] 온톨로지 저작 도구 비교 ····· 13
[표 8] 구글과 야후의 검색 결과 ····· 16
[표 9] 워드넷 심볼과 OWL 어휘 맵핑 테이블 ····· 19
[표 10] 자동 온톨로지 구축 과정 ····· 19
[표 11] OWL 어휘들의 고유관계 ····· 26
[표 12] JESS의 추론규칙 ····· 27
[표 13] 대표적 추론 규칙과 적용되는 OWL 어휘들 ····· 31
[표 14] 추론 규칙에 대한 트리플 변환 모듈 ····· 32
[표 15] 온톨로지 질의 언어 비교 ····· 38
[표 16] 질의 처리 모듈의 특징 ····· 38
[표 17] 질의 수행 방식 ····· 42
[표 18] 질의 수행 과정 상세 ····· 43
[표 19] 온톨로지 저작 도구 비교 ····· 48
[표 20] Jena의 RDF 온톨로지 파싱 모듈 ····· 49
[표 21] 본 연구에서 제안한 온톨로지 파싱 모듈 ····· 50
[표 22] 온톨로지 저작 도구의 파싱 속도 ····· 51
[표 23] 질의 처리 결과 비교 ····· 55

도 목 차

[그림 1] 시맨틱 웹 기술 ····· 6
[그림 2] 시맨틱 웹 구조 ····· 7
[그림 3] Protege 실행화면 ····· 11
[그림 4] OntoEdit 실행화면 ····· 12
[그림 5] OWL Editor 실행화면 ····· 13
[그림 6] 온톨로지 기반 의미적 정보 검색 시스템 ····· 14
[그림 7] 워드넷 테이블 구조 ····· 17
[그림 8] 워드넷 계층 구조 ····· 18
[그림 9] 워드넷 기반 자동 온톨로지 구축 내용 ····· 22
[그림 10] 온톨로지 자동 구축 프로세스 ····· 23
[그림 11] 자동 온톨로지 구축 결과 비교 ····· 24
[그림 12] 파서 처리 과정 ····· 29
[그림 13] Blank Node 처리 ····· 30
[그림 14] 전이-'rdfs:subClassOf' ····· 33
[그림 15] OWL 1차 Syntax 파싱 예 ····· 35
[그림 16] OWL 2차 Semantic 파싱 예 ····· 35
[그림 17] OWL 어휘기반 Fact 트리플 생성 ····· 36
[그림 18] 추론 규칙 적용 Semantic 트리플 생성 ····· 36
[그림 19] 질의 처리 모듈의 전체적인 수행 프로세스 ····· 46
[그림 20] 질의 처리 모듈 실행 화면 ····· 47
[그림 21] 온톨로지 저작 도구의 단위 시간당 트리플 처리량 ····· 51
[그림 22] SPARQLer의 질의 수행 인터페이스 화면 ····· 52
[그림 23] SPARQLer의 질의 수행 결과 값 ····· 53
[그림 24] 본 연구의 질의 인터페이스 화면 ····· 53

ABSTRACT

The Method of Automatic Ontology Building and Efficient Query Processing for the Semantic Information Retrieval

Kong, Hyun-Jang
Advisor: Prof. Kim, Pan-Koo, Ph. D.
Department of Computer Science
Graduate School of Chosun University

As the study on the semantic web has been progressed, the applicable range of the ontology is getting wide. So, lots of ontologies were built and are exiting on the web nowadays. As many ontologies are made, several problems occured on the ontology side as well as on the semantic web.

Ontology plays a primary role in the semantic information retrieval and several methodologies are needed to process the ontology efficiently. In this thesis, the automatic ontology building method, efficient ontology parsing method and triple-based ontology query processing method were studied. Firstly, the automatic ontology building method is designed based on the WordNet. The concepts collection, class creation and relation creation between classes among the ontology development steps are processed automatically. Secondly, the ontology parser is newly designed for the efficient ontology parsing. As a result, the parsing time was improved. The syntax parsing and semantic parsing about ontology are processed, and then two types of triple

set are created throughout the parser. Finally, created triple sets are used as the input data of the ontology query processing method suggested. Therefore, the query processing method supports the semantic information retrieval about the ontology and user can write the queries easily in the query interface using the controlled query process module.

Although the active study on the semantic web brings about the implementation of the ontology techniques, several methods for processing the ontologies efficiently are still demand. It should be studied about how to get the rich knowledge data from the domain experts or the others for the automatic ontology building and apply the query processing method for the ontologies to the information retrieval system.

제 1 장 서 론

제 1 절 연구 배경 및 필요성

네트워크 인프라와 웹 기술의 성장은 누구나 쉽게 원하는 정보를 접근하고 활용할 수 있는 환경을 제공한다. 현재의 웹은 HTML을 사용하여 정보를 표현한다. 이러한 환경에서는 사용자가 원하는 정보를 기계가 스스로 이해하고 처리하기 위해 여러 제약이 따른다. 그리하여 웹 정보를 컴퓨터가 이해하고 인간과 흡사한 정보처리를 수행 할 수 있는 시맨틱 웹 환경이 대두되었다. 시맨틱 웹 환경에서는 웹의 정보에 인간의 사고방식을 반영한 잘 정의된 의미(Semantic)를 부여한다. 이러한 의미에 대하여 인간의 사고방식과 흡사한 추론 기능이 추가되어 웹은 디스플레이 중심에서 의미중심으로 변화한다.

이러한 시맨틱 웹의 핵심 기술이 바로 온톨로지(Ontology)이다. 온톨로지는 개념(Concept)들의 구조를 상세화하는 것으로 개념들 사이에 존재하는 관계를 명백하게 기술한다. 온톨로지는 "공유(shared)를 위한 개념들의 개념화(conceptualization)를 형식적(formal)이고, 명백하게(explicit) 설명해 놓은 명세서(specification)"라고 정의 할 수 있다[1, 2]. 정의에서 '개념화'는 어떤 현상과 관련있는 개념들을 식별하여 그 현상을 추상화한 모델로 표현하는 것이며, '명백하게 표현한다'는 의미는 사용된 개념들의 종류와 개념들이 갖는 관계들과 개념들의 사용 시 고려되어야 하는 제약 사항들을 명확하게 정의한다는 것이다. '형식적인 표현'은 온톨로지가 기계 판독이 가능한 형태여야 한다는 것을 의미하고, '공유된 개념'은 개별적이거나 사적인 용도의 온톨로지뿐만 아니라 집단의 의견 수렴을 통해 집단의 일관된 지식을 정의하고 있음을 의미한다.

온톨로지에 대한 연구는 온톨로지의 구축 방법론, 온톨로지 구축 언어, 온톨로지 저작 도구, 온톨로지 질의 처리 및 추론 엔진에 대한 연구에 이르기까지 다양한 분야에서 진행되어지고 있다. 본 논문에서는 이러한 온톨로지에 대한 연구 중에서도 특히, 향후 시맨틱 웹 환경에서 의미적 정보 검색을 위해 요구되는 효율적

온톨로지 처리 기술들 중 온톨로지의 재사용을 위한 자동 구축 방법과 온톨로지의 효율적 평가를 위한 질의 처리 기술에 대하여 연구 하고자 한다.

제 2 절 연구 내용 및 범위

온톨로지가 시맨틱 웹의 핵심이 되기 위해서 온톨로지 설계 및 구축을 담당하는 저작도구를 개발하고, 이를 통해 구축된 온톨로지에서 유용한 정보를 추출하는 연구가 필요하다. 본 연구에서는 온톨로지를 효율적으로 처리하기 위해 다음의 세 가지 기술을 연구한다.

① 기존 지식의 재활용을 위한 자동 온톨로지 구축 방법
② 온톨로지 파서의 성능을 개선한 트리플 온톨로지 파싱 모듈 설계
③ 트리플 기반의 질의 처리를 통한 효율적 온톨로지 내 정보 탐색 방법

본 논문의 2장에서는 연구의 기반이 되어진 시맨틱 웹, 온톨로지, 그리고 기존 온톨로지 저작도구들에 관한 관련 연구를 기술하고, 3장에서는 본 연구의 핵심인 온톨로지 자동 구축 모듈과 파싱 모듈 및 효율적 온톨로지 질의 처리를 통한 온톨로지 내 정보 탐색에 대한 연구 내용을 기술한다. 4장에서는 각각의 과정에서 얻어진 실험 결과를 토대로 본 연구의 우수성을 제시하고, 5장에서 결론을 맺는다.

제 2 장 관련연구

제 1 절 온톨로지와 시맨틱 웹

1. 온톨로지(Ontology)

온톨로지란 어휘나 개념의 정의 또는 명세로서 정보시스템 분야에서는 시스템이 다루는 내용에 해당한다. 또 온톨로지는 데이터베이스의 일종이라 할 수 있는데, 보통의 관계형 데이터베이스와 달리 개념들 간 관계와 개념에 대한 제약이 표현되어 있다. 온톨로지는 가장 단순한 형태로 어휘사전이나 용어모음을 생각할 수 있지만 컴퓨터가 처리할 수 있을 만큼의 구조성과 구체성을 갖춰야 온톨로지라 할 수 있다. 온톨로지에 대한 연구는 지식 표현 기술과 관련된 중요한 분야로서 시맨틱 웹과는 독립적으로 발전해 왔다. 이와는 대조적으로 시맨틱 웹은 하나의 독립된 기술이라기보다는 RDF와 같은 메타데이터 언어를 비롯하여 여러 분야의 기술들이 연합하여 발전되어 온 기술로서, 온톨로지 기술은 시맨틱 웹 구현을 위한 중요한 요소가 되었다. 따라서 온톨로지에 대한 연구는 더욱 활기를 띠게 되었으며 응용범위도 갈수록 확산되고 있다[5, 6, 7, 8].

최근에는 여러 가지 방법으로 온톨로지를 구축하고자하는 방법론이 제시되고 있다. OTK(On-To-Knowledge) 방법론은 응용프로그램 지향의 온톨로지 개발에 중점을 두고, 사용자관점과 소프트웨어 엔지니어링 관점을 동시에 고려한 모델이다[29].

[표 1] OTK 방법론

단 계	내 용
Ontology 활용 타당성 조사	- 문제점 및 가능성 확인 - 지식관리 응용 프로그램 초점 - 도구 및 포함 대상 확인
Ontology 모델링 시작	- 요구명세(OSRD) 작성 - 지식정보원 분석 - 초기 Ontology 개발
Ontology 정제	- 초기 Ontology 검증 및 정제 - 공식 Ontology 생성
Ontology 평가	- 기술 관점 평가 (온톨로지의 구문적 정합성 및 의미적 일관성) - 사용자 관점 평가 (온톨로지가 ORDS에 명세된 정보를 모두 포함하고 있는지 점검. 필요없는 부분은 없는지 검토) - 온톨로지 구조 관점 평가 (온톨로지의 의미적 정합성 점검. 잘못된 개념화 여부를 찾기 위해 OntoClean과 같은 방법 혹은 온톨로지 평가 규칙 이용 가능)
서비스 시스템 구축/평가	- 계속적인 유지, 개선과정

또 하나의 대표적 온톨로지 구축 방법론은 Ontology Development 101 이다. 이는 Protege를 개발한 스탠포드 대학의 지식시스템연구소에서 제시한 것이다[28, 30]. 요약하면 [표 2]와 같다.

[표 2] Ontology Development 101 방법론

단 계	내 용
적용 범위 결정	- Ontology의 영역과 범위 결정
기존 자원 재활용 검토	- 기존 Ontology 활용 가능성 검토
용어를 열거	- Ontology에 사용될 중요한 단어 열거
개념(Class) 정의	- Class와 Class의 계층 구조 정의
속성 정의	- Class의 Property(Slot) 정의
제약조건 정의	- Property의 제약사항 정의
Instance 생성	- Class의 Property에 대한 값을 채움

앞에서 소개하는 두 가지의 온톨로지 구축 방법론을 보면, 온톨로지를 구축하는데 여러 단계의 과정을 거쳐야 한다. 특히, 이러한 과정들 중에서 온톨로지의 “기존 자원의 재활용과 온톨로지 평가” 항목은 온톨로지 구축에 있어서 중요한 부분이다. 하지만 기존 온톨로지 구축 시 고려대상에서 제외되어 왔다. 본 연구에서는 온톨로지 구축 방법론에서 제시한 온톨로지 평가와 기존 자원의 재활용 검토라는 부분을 만족하도록 온톨로지 처리 기술들을 개발하였다.

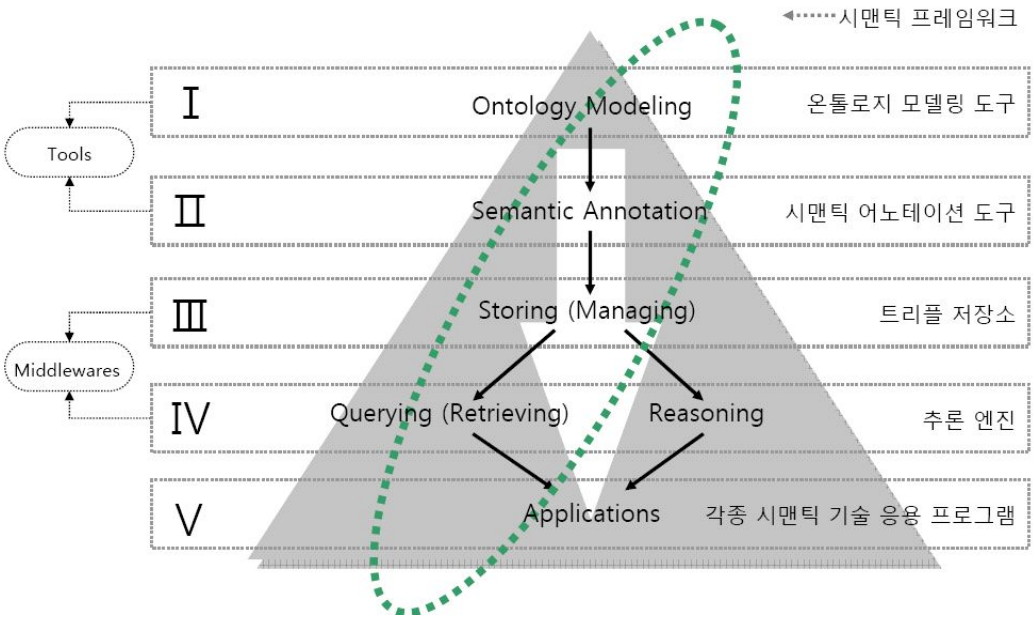
2. 시맨틱 웹

1994년 10월 팀 버너스리는 월드와이드웹을 발표하였고, 웹이 발표되고 10여년의 세월이 흐른 후, 시맨틱 웹을 차기작으로 발표하였다. 팀 버너스리는 최근 월드와이드웹 컨퍼런스에서 “시맨틱 웹이 2 단계로 진화하고 있다”라고 말했다. 1단계에서 거둔 커다란 성과는 RDF, OWL과 같은 표준 언어를 만든 것이며, 2단계는 실제 애플리케이션을 개발하고 응용하는 단계라 말할 수 있다. 팀 버너스리가 처음 웹을 발표했을 때 웹은 아직 완성되지 않은 것이었다. 팀 버너스리가 제시한 시맨틱 웹은 ‘컴퓨터가 정보의 의미를 이해하고 조작할 수 있는 웹’으로 정의 할 수 있다. 여기서 말하는 ‘의미를 이해한다’고 하는 것은 ‘이해의 대상이 되는 새로운 것을 이미 알고 있는 다른 것과 관계 짓는 능력’을 말한다.

[표 3] 기존 웹 기술과 시맨틱 웹 기술 특성 비교

항 목	기존 웹 특성	시맨틱 웹 특성
Data Modeling	ER Model	Ontological Model
Schema Territory	Syntax	Semantics
Data Structure	Table	Graph
Data Element	2-Tuple	Triple
Storing, Managing	DBMS	KB(Triple Store)
Query Language	SQL	SPARQL
Processing(Programming)	Procedure(Java,C...)	Declarative(Prolog,Lisp...)
Reasoning	Hard Coding	Logic Rule
Service Area	Information	Knowledge

[표 3]에서와 같이 시맨틱 웹 기술은 기존 웹의 대부분을 차지하는 데이터베이스 기반의 기술과 차이가 있다. 이러한 시맨틱 웹 기술들이 실현되어야 진정한 시맨틱 웹 환경이 완성되었다고 할 수 있다. 시맨틱 웹을 위한 연구는 [그림 1]의 내용에 기반하여 진행되어지고 있다.

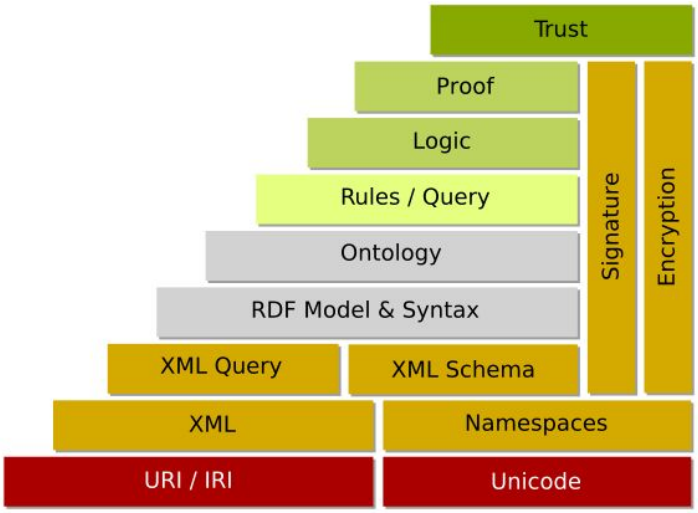


[그림 1] 시맨틱 웹 기술

[그림 1]에서도 확인할 수 있듯이, 시맨틱 웹의 구현을 위해 온톨로지 구축과 구축된 온톨로지의 표현과 저장 그리고 온톨로지에 대한 추론 모듈 마지막으로 이러한 온톨로지를 활용하기 위한 응용 프로그램들이 개발되어야 하는 등의 여러 기술들이 복합적으로 요구된다.

현재의 웹상에는 자료가 널려있을 뿐이고 이것이 어떤 정보인지에 관한 판단은 모두 인간이 한다. 이런 자료를 구조적으로 정리하고 분류하게 되면 이것은 정보가 되며, 이 정보들을 바탕으로 추론된 결과가 바로 지식(Knowledge)이다. 시맨틱 웹은 단순한 자료로만 존재하던 웹을 구조적인 분류와 정보들 사이의 추론이 가능하도록 바꾸고자 한다. 기존의 웹에서는 사람이 자료를 보고 나름대로 추론해서 자료를 정보화했다면, 시맨틱 웹에서는 이런 기능을 에이전트라고 부르는

어플리케이션이 대신한다. 시맨틱 웹의 연구는 아직 진행 중이며, 인터넷은 분명 변하고 있다. 정적이었던 인터넷은 이제 점점 동적 모습으로 변해가고 있다. 문서의 각 부분을 컴퓨터가 이해할 수 있는 형식으로 기술(description)해 줄 수만 있다면 복잡하게 얽혀져 있는 정보 자원들 사이의 의미적 연관성을 통해 다양한 정보들이 보다 효과적으로 활용될 수 있다. 이러한 시맨틱 웹의 미래를 위해 다양한 팀 버너스리는 시맨틱 웹 구조를 제시하였다[5].



[그림 2] 시맨틱 웹 구조

시맨틱 웹의 계층적 구조를 살펴보면 [그림 2]와 같다. 첫 번째 중요한 구성요소가 바로 XML이다. HTML이 웹 문서를 화면에 보여주기 위한 언어라면 XML은 웹에서의 데이터를 기술하기 위한 언어다. XML에서 정의 내려지는 태그의 의미는 사용자가 이해하기 쉽게 사용자가 결정할 수 있다. 그러나 XML은 문서 요소들 사이의 구조적 정의만을 제공하기 때문에 정보들 사이의 의미적 관계를 정의 할 수 없다. 두 번째 요소는 RDF 와 RDF 스키마이다. RDF 문서의 문법은 XML에 기반하고 있으며, RDF는 객체 지향적 접근의 지식표현 방식으로 세 개의 요소로 이루어졌다. 객체(object)-속성(attribute)-값(value)의 구조로 이루어져 있으며, A(O, V)로 표현된다. 그러나 일반적인 객체 지향적 혹은 프레임 방식과는

달리 RDF는 객체 중심이 아니라 속성 중심(property-centered)의 구조를 가지고 있으며, RDF 데이터 모형은 방향성 있는 그래프의 구조로 나타내어진다. RDF 트리플 표현 방식의 예는 [표 4]와 같다[4].

[표 4] RDF 표현

```
<rdf:Description about='http://www.bookstore.org/id1971'>
<author>'http:// www.seoul.ac.kr/kdhong'</author>
</rdf:Description>
```

RDF 스키마는 XML 문서의 요소들 사이의 구조를 정의해 주는 XML 스키마와 달리, RDF 문장에서 쓰여진 'author,' 'title,' 'book'과 같은 어휘들의 정의를 위한 메타언어이다. RDF 스키마에는 미리 정의된 몇 개의 어휘들이 RDF 문장에서 쓰이는 어휘들 간의 관계를 의미적으로 정의한다. 예를 들면, RDF 문장에서 각각의 개체(instance)는 유형(type) 속성을 가지는 하나 이상의 클래스의 원소(element)로 정의 내려진다. 또한 subClassOf 속성은 클래스들 사이의 계층적 구조를 나타내는데 사용된다. 예를 들어, 소설책 클래스와 책 클래스 사이의 관계를 RDF 스키마로 표현하면 [표 5]와 같다.

[표 5] RDF 스키마 표현

```
<rdfs:Class rdf:ID='Novel'>
        <rdfs:subClassOf rdf:resource='#Book'/>
</rdfs:Class>
```

[표 5]에서 속성 title을 RDF 스키마로 정의하면 [표 6]과 같다.

[표 6] RDF 스키마를 사용한 Title 정의

```
<rdf:Property ID='title'>
            <rdfs:domain rdf:resource='#Book'/>
<rdfs:range rdf:resource='http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Literal'/>
</rdf:Property>
```

[표 6]에서 속성 title은 Book을 도메인으로 Literal을 치역(range)으로 갖는 대상으로 정의된다.

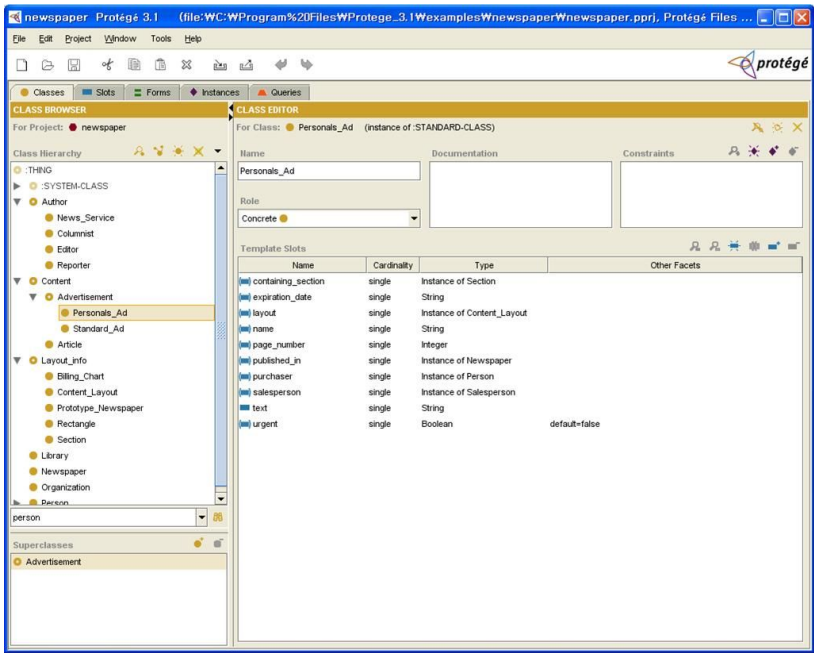
세 번째 구성요소가 바로 온톨로지이다. 온톨로지는 앞서 설명하였듯이 기계 또는 에이전트가 처리할 수 있는 풍부한 어휘와 형식적 의미를 제공하여 어떤 용어의 의미와 용어 사이의 관계를 명시적으로 표현한다. 그리고 로직은 온톨로지를 지원하는 추론역할을 하는 부분이며, 증명과 사실 부분은 현재까지 많은 연구가 진행되지 않은 제안부분이다.

제 2 절 국내외 온톨로지 저작 도구

1. 기존 온톨로지 저작 도구 비교·분석

온톨로지 처리 기술에 대한 현재의 기술들은 기 구축된 온톨로지 저작 도구에 반영되어 있다. 기존의 국내외 온톨로지 저작 도구들을 조사·분석하여 현재까지의 온톨로지 처리 기술들에 대하여 파악한다. 국외의 대표적 시스템인 KAON, Protege 3.1, OilEd, OntoEdit, SMORE, Ontolingua를 비롯하여, 국내 온톨로지 저작 도구인 OWL Editor와 OWL 그래픽 편집기에 대하여 조사·분석한 내용은 다음과 같다[20, 29, 30].

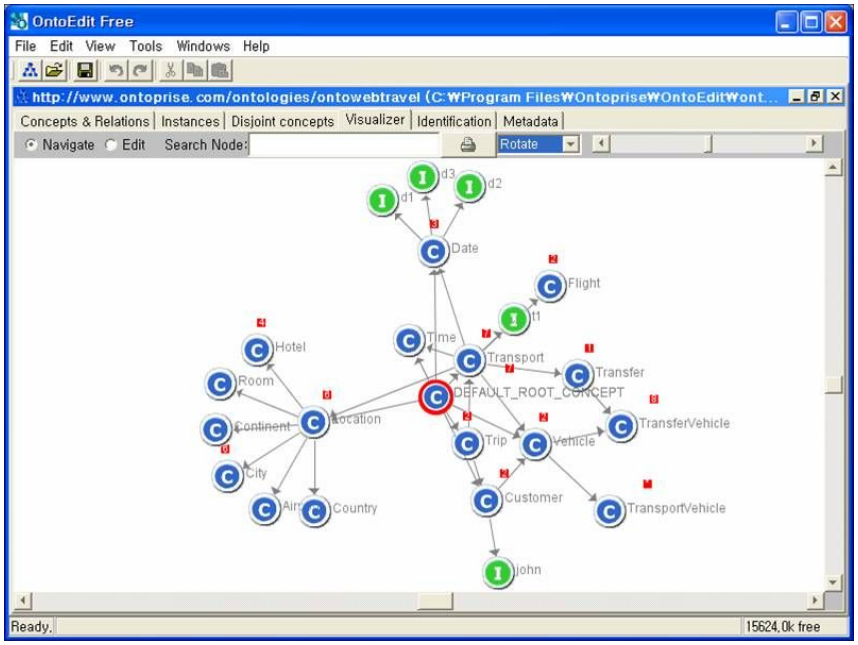
독일의 KArlsruhe 대학과 FZI 연구회에서 공동으로 제작한 KAON은 웹 기반으로 구축되어졌으며, 서버에 온톨로지들을 저장하고 여러 사용자가 동시에 접속하여 온톨로지를 생성하고 편집할 수 있다. 또한 그래프 기반으로 온톨로지를 표현하여 쉬운 조작을 통한 자원(클래스, 속성, 인스턴스)들의 관리가 가능하다. KAON은 고유의 프로젝트 파일을 생성하여 온톨로지 모델을 구성하는 방법을 사용한다. 미국의 스탠포드 대학에서 제작한 Protege 3.1은 현재 온톨로지 구축에서 가장 널리 활용되고 있다. 트리 구조를 이용하여 자원들의 상/하위 계층 구조를 표현함으로써 쉽게 자원의 정보를 확인하고 수정할 수 있다. 또한 자원을 정의할 때, 폼(Form) 인터페이스를 사용하여 클래스와 속성을 정의함으로써, 자동으로 인스턴스를 정의할 수 있는 폼이 구성되어 온톨로지 구축이 용이하다. 특히, Protege 3.1이 가장 많이 활용되고 있는 이유는 확장 가능한 플러그인 구조로 제작되어, 공통의 라이브러리를 이용하여 온톨로지 저작 도구를 개발할 수 있기 때문이다. OilEd는 영국의 맨체스터 대학에서 DAML+OIL언어에 기반하여 제작한 온톨로지 편집기로써, 다른 온톨로지 도구에 비해 쉬운 인터페이스로 구성된 점이 특징이다. OilEd의 주요 기능은 FaCT 추론기를 이용하여 온톨로지들을 분석하고, DAML+OIL을 SHIQ Description Logic으로 변환하여 온톨로지의 일관성을 체크할 수 있다[9].



[그림 3] Protege 실행화면

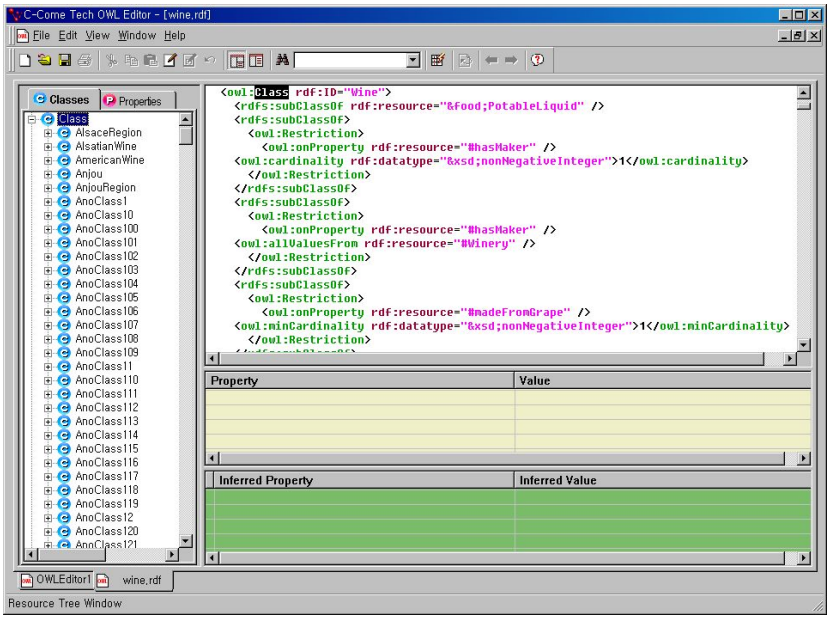
독일의 Ontoprise에서 개발한 OntoEdit는 OilEd와 동일하게 DAML+OIL에 기반하고 있으며, 그래픽 사용자 인터페이스를 제공하지만 조작은 조금 복잡하다. 트리로 구성된 계층 구조 내 Copy and Paste가 가능하여 개념들을 쉽게 재조직할 수 있는 특징이 있다. 또, 구성된 개념들은 Visualization을 통해 그래프로 표현하여 쉽게 온톨로지의 전체 구조를 파악할 수 있고, 그래프 내에서의 편집기능도 제공한다. 그리고 Protege 3.1처럼 플러그인 구조를 제공하여 필요한 기능들을 쉽게 확장할 수 있는 장점이 있다. 온톨로지 저작 도구인 SMORE(Semantic Markup, Ontology and RDF Editor)는 Java로 구현되었다. WYSIWYG HTML 편집기와 온톨로지를 검색할 수 있는 웹 브라우저 역할을 하며, GUI(Graphic User Interface)를 이용하여 온톨로지를 생성하고 관리할 수 있는 기능을 가지고 있다. Ontolingua는 웹 상에 분산되어 있는 온톨로지를 사용자가 생성하고 관리할 수 있는 시스템으로 두 가지의 중요한 기능을 가지고 있다. 여러 개의 온톨로지들을 통합할 수 있으며, 온톨로지를 해석할 수 있는 기능이 있다. Ontolingua는 지식베이스를 각기 다른 형태로 가져와서 분류하고 재구성하여 이름 충돌 문제를

해결하고 온톨로지를 브라우징하는 기능과 용어를 편집하는 기능 등도 포함하고 있다.



[그림 4] OntoEdit 실행화면

국내 기업인 씨컴테크 사에서 개발된 OWL Editor는 순수 OWL 파일만을 처리하고 있으며, 텍스트 편집창 형식의 인터페이스를 제공한다. 그렇지만 자원의 추가, 수정 및 삭제 등의 편집은 다이얼로그 창을 띄워 이루어진다. OWL Editor는 텍스트 편집창 방식으로 구성되어져 있기 때문에 Serialization 기능이 따로 존재하지 않으며, 온톨로지의 파싱과 저장이 빠르고, 처리 가능한 데이터양도 다른 시스템들에 비해 큰 편이다. 또한 여러 온톨로지를 동시에 읽어 들여, 편집이 가능한 특징이 있다. OWL 그래픽 편집기도 국내 기업인 케이테크 사에서 개발하였으며, 그래프 표현 방식을 이용하여 온톨로지를 모델링하고, DAML+OIL과 OWL파일을 시각화하여 다룰 수 있다. 그리고 RDF(S), DAML+OIL, OWL로 작성된 문서의 파싱과 Serialization 기능을 지원한다.



[그림 5] OWL Editor 실행화면

앞서 조사한 다양한 구축 도구들에서 알 수 있듯이, 온톨로지에 대한 연구는 활발하게 진행되고 있다. [표 7]은 조사・분석한 온톨로지 저작 도구들의 특징을 비교한 내용이다.

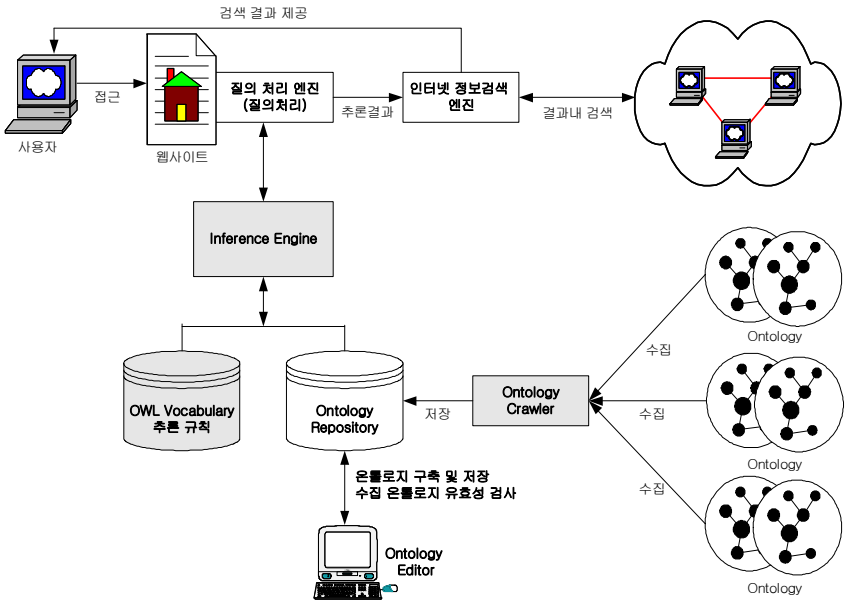
[표 7] 온톨로지 저작 도구 비교

저작 도구	편집 방식	파일 형태	특 징
KAON	Graph	(.kws)Project File	웹기반 멀티 구축 지원
Protege 3.0	Tree & Form	(.pprj)Project File	다양한 plug_in 지원
OilEd	Tree & Form	DAML+OIL,OWL	일관성 체크 지원
OntoEdit	Tree & Graph	DAML+OIL	Visualization 지원
OWL Editor	Text	OWL	텍스트 편집 창 방식

제 3 장 효율적인 온톨로지 처리

제 1 절 온톨로지 기반 의미적 정보검색

지금까지 온톨로지 기반 정보검색 시스템[16, 17, 21, 23] 연구에서는 온톨로지를 적용하여 의미적인 정보검색을 시도하였다. 하지만 온톨로지 기반의 정보검색 시스템을 구현하기 위해서는 온톨로지의 구축에서부터 온톨로지의 관리 그리고 온톨로지를 이용한 추론을 비롯하여 온톨로지 기반의 질의 처리 방법 등 여러 가지 요소 기술들을 요구된다. 본 연구에서 제안하는 의미적 정보검색 시스템에서 온톨로지는 핵심 역할을 담당하고 있으며, 의미적 정보검색을 위해 온톨로지를 효율적으로 처리하기 위한 요소 기술들에 대하여 연구하였다. 본 연구에서는 의미적 정보검색이 가능하도록 온톨로지 관련 요소 기술들을 새롭게 설계하였다. 온톨로지 기반의 의미적 정보검색 시스템의 구축을 위해 세 가지의 온톨로지에 관련된 주요 기술들이 필요하며, 이에 기반하여 제안하는 의미적 정보검색 시스템의 구조는 [그림 6]과 같다.



[그림 6] 온톨로지 기반 의미적 정보검색 시스템

[그림 6]에서 요구되는 첫 번째 기술은 온톨로지 구축 도구이다. 사용자는 구축 도구를 사용하여 온톨로지의 구축 및 관리를 수행한다. 두 번째로 추론 엔진에서는 온톨로지를 JESS 추론 규칙에 기반하여 파싱을 하여 의미적 트리플로 구성한다. 이렇게 작성된 의미적 트리플은 세 번째 기술인 온톨로지 질의 처리에 입력 데이터로 사용된다. 온톨로지 질의 처리에서는 사용자 중심의 질의 인터페이스 및 추론된 내용에 대한 질의를 수행하여, 온톨로지 내 의미적 정보검색이 가능하다. [그림 6]은 본 연구에서 제안하는 온톨로지 기반 정보검색 시스템의 구조를 설명하고 있으며, 여기에는 세 가지의 요소 기술들이 유기적으로 결합되어 전체 시스템이 구성되어져 있다. 시스템은 기존의 웹과 접목되어 웹의 정보에 대하여 의미적 정보검색이 가능하도록 설계되었다. 전체 시스템의 흐름을 순서적으로 살펴보면 다음과 같다.

1. 사용자 : 정보검색 엔진에 접근
2. 추론엔진에서 지원하는 트리플 사실들을 이용한 사용자 질의 입력
3. 질의에 대하여 온톨로지 저장소에 저장된 온톨로지를 사용하여 특정 추론 규칙을 적용한 추론 엔진에 의해 추론된 사실들을 기반으로 해당 결과값을 얻음
4. 온톨로지 저작도구는 온톨로지를 새롭게 구축하고 관리함
5. 온톨로지 기반의 검색 결과값을 검색 엔진에 전달하고, 웹 검색 엔진은 검색 결과에 기반한 웹 정보검색 시도

온톨로지를 활용한 웹에서의 의미적 정보검색을 위해 많은 기술들이 요구되며, 특히 온톨로지에 관련된 기술이 큰 비중을 차지하고 있다. 본 연구에서는 온톨로지 기반의 정보검색을 위한 전체적인 시스템 설계와 세부적으로 필요한 요소 기술들을 개발하여 온톨로지 기반의 의미적 인터넷 정보검색이 가능함을 보인다. 본 연구에서는 온톨로지를 적용한 의미적 인터넷 정보검색 위해 온톨로지 자동 구축 방법, 파싱방법 그리고 효율적 질의 처리 방법 등을 설계하여 전체적인 의미적 정보검색 시스템을 구성하였다.

본 연구에서 제안하는 시스템이 기존의 웹에서의 정보 검색과는 차별화된 검색 결과를 제공함을 제시하기 위해 [표 8]의 네 가지 검색 질의에 대하여 현재 웹의 대표적 검색 엔진인 구글과 야후에서 검색을 수행하여 결과를 확인하였다.

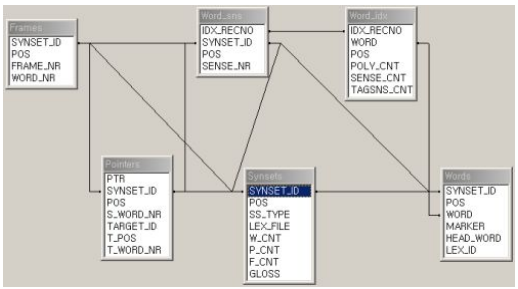
[표 8] 구글과 야후의 검색 결과

	구글(Google)	야후(Yahoo)
오찬 와인 (가볍고 드라이한 와인)	41/100	35/100
조개류 (드라이 화이트 와인)	53/100	47/100
토프 스테이크 (단맛이 강한 화이트 진판델)	8/100	11/100
매운 음식에 어울리는 와인 (단맛이 약한 화이트 와인 독일 알자스 지방 와인)	16/100	17/100

[표 8]의 결과는 각각의 검색엔진에서 검색한 전체 결과 중에 최상위 100개를 대상으로 해당 질의에 대한 정확한 결과를 나타내고 있다. [표 8]의 실험 결과에서 확인할 수 있듯이, 기존의 인터넷 정보 검색 환경에서의 정보 검색은 정보에 대한 질적인 검색보다 양적인 검색을 수행해 왔다. 이러한 근본적인 이유가 바로 기존의 웹 구조에서는 정보의 효율적 가공이 불가능하기 때문이다. 이러한 점에서 요구되는 것이 정보의 효율적 가공 기술인 온톨로지 기술이며, 온톨로지를 효율적으로 처리하는 기술이 의미적 정보검색에서 핵심의 부분을 차지한다. 이에 본 연구에서는 온톨로지 기반의 의미적인 인터넷 정보검색을 위한 효율적 온톨로지 처리 기술들을 개발하였다.

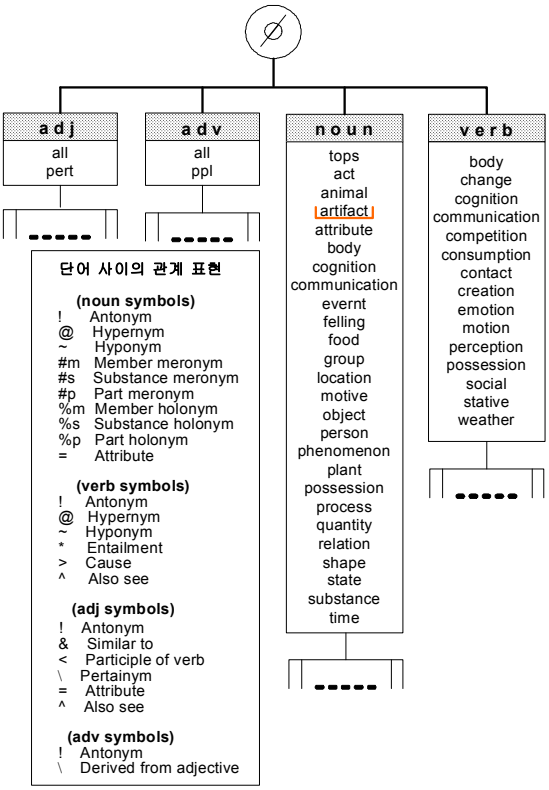
제 2 절 온톨로지 구축

온톨로지의 중요한 특징 중의 하나는 기존에 구축된 온톨로지 또는 체계적인 지식을 공유하고 재사용하는 것이다. 본 연구에서는 효율적 온톨로지 처리를 위해 워드넷 기반 온톨로지 자동 구축 방법에 대하여 연구한다. 본 연구의 온톨로지 자동 구축 방법은 개념 수집과 개념간의 관계 설정이 자동으로 이루어지도록 지원한다. 자동 온톨로지 구축 방법은 워드넷에 기반하고 있으며, 워드넷[18, 19, 22]은 범용의 대형 온톨로지로써 그 내용은 6개의 데이터베이스 테이블들로 구성되어져 실세계에 존재하는 어휘에 대해서 체계적으로 정의하고 있다. 워드넷에서 중요한 내용은 바로 개념간의 관계를 정의하고 있는 부분이며, 사전과 가장 큰 차이점이 또한 바로 이러한 부분이다.



[그림 7] 워드넷 테이블 구조

워드넷은 크게 4개의 카테고리(명사, 형용사, 부사, 동사)로 분류되고, 내부적으로 45개의 서브카테고리로 분류된다. 그리고 워드넷 내의 모든 개념들은 특정 심벌들을 사용하여 각 개념들 간의 관계를 표현하고 있다. [그림 7]과 [그림 8]은 워드넷의 구조를 표현하고 있다.



[그림 8] 워드넷 계층 구조

워드넷은 개념들 간의 관계를 체계적으로 정의한 범용 온톨로지이지만 워드넷은 데이터베이스 기반으로 구성되어져 있어 모든 처리는 데이터베이스 기반 특정 질의와 데이터베이스 기반 특정 어플리케이션을 통하여서만 이루어진다. 그래서 워드넷을 시맨틱 웹 환경에 적용하기에 많은 제약이 있다. 그리하여 본 연구에서는 이러한 데이터베이스 기반의 워드넷 내용을 분석하여 OWL 온톨로지로 재구성하기 위한 자동 온톨로지 방법에 대하여 연구하였다.

워드넷에 기반 한 자동 온톨로지 구축 방법은 워드넷 내의 6개의 테이블 중에서 Pointers, Synsets, Words 테이블을 사용하고 있다. 또, 워드넷에서 정의하고 있는 명사, 동사, 형용사, 부사 중에서 명사 부분을 자동 구축 방법에서 활용한다. 자동 구축에는 워드넷 내 명사들 간의 관계를 정의하는 워드넷 심벌들을 파악하여, 이를 OWL 어휘들과 맵핑하기 위한 맵핑 테이블을 구성하였다.

[표 9] 워드넷 심벌과 OWL 어휘 맵핑 테이블

워드넷	OWL 어휘
!	owl:disjointWith
@	owl:superClassOf
~	owl:subClassOf
동일 synset ID	owl:equivalentClass

온톨로지 자동 구축 방법의 과정에서 워드넷의 Words 테이블에서 추출된 명사들은 OWL에서의 클래스들로 각각 맵핑되고 각 심벌들은 [표 9]의 맵핑 정보에 따라 OWL 어휘들로 재구성된다. 워드넷에 기반 한 자동 온톨로지 구축 방법의 전체 알고리즘은 다음과 같다.

[표 10] 자동 온톨로지 구축 과정

단 계	내 용
STEP 1	워드넷 내의 개념 중 명사부분 추출
STEP 2	개념들간의 관계(상하위/동의/반의) 설정
STEP 3	개념과 관계의 OWL 변환
STEP 4	트리플 파싱을 통한 온톨로지 작성

STEP 1 : 워드넷의 개념들 중 명사를 추출한다.

내 용 : 워드넷의 개념들 중 명사는 words 테이블의 POS 필드가 'n'인 개념들만을 추출하면 된다.

모 듈 :

```
select synset_id, word, pos, lex_id
from Table(words)
where pos = 'n'
```

설 명 : 여기에서, 선택되는 필드값 중 sysset_id는 primary 값으로 모든 테이블과 연결하기 위하여 사용되고, word는 시스템에서 사용자 인터페이스 시 보여주기 위한 값으로 추출된다. 또한 lex_id 값은 워드넷 내에 모든 개념들은 lex_id로 구룹화 되어져 있어, 해당 개념이 호출되었을 때, 그 개념의 lex_id 값을 기반으로 개념들을 탐색하면 관련된 개념들을 더 빠르게 탐색하여, 자동 온톨로지 구축의 시간을 단축할 수 있다.

STEP 2 : 상하위/동의/반의 관계를 찾는다.

내 용 : 워드넷의 개념들 중 명사가 갖고 있는 대표적인 개념들 간의 관계가 바로 상하위/동의/반의의 관계이다. 이러한 관계를 추출하기 위하여 워드넷의 pointers 테이블의 PTR 필드의 값을 사용한다. 여기에서 PTR의 값은 개념들 사이의 관계를 표현하고 있으며, 상위관계는 '@'로 하위관계는 '~' 그리고 반의 관계는 '!'으로 표현이 되면, 예외적으로 동의관계는 동일 synset_id로 정의된다. 이러한 개념들 간의 관계는 다음과 같이 추출될 있다.

모 듈 :

```
select target_id, synset_id, ptr, pos
from Table(pointers)
where synset_id in (Step 1의 synset_id)
```

설　명 : 여기에서, synset_id에 기반해서 해당 synset_id가 가지고 있는 모든 ptr 값을 추출한다. 이러한 ptr 값에는 각각 한 개의 target_id가 매칭되며, 해당 targer_id는 다시 Step 1의 synset_id 대신에 조건절에 들어가서 원래 단어와 ptr 관계의 개념을 찾을 수 있다. 역시, pos 값을 참고로하여, pos가 'n' 인 것만을 사용함으로써, Step 1에서와 마찬가지로 시간적인 절약이 가능하다.

Step 2는 해당하는 개념에 대하여 ptr 값이 있는 동안 반복하여 수행되어, 해당 개념과 관련된 모든 개념들을 추출한다.

STEP 3 : 개념들과 관계들은 온톨로지에 적합하게 변형된다.

내　용 : 온톨로지 구축을 위해 개념들은 Class로 관계들은 Relation으로 표현된다. 여기에서, 사용되는 구조가 바로 triple 구조이다. 가장 먼저 들어오는 값은 Step 1에서 얻어진 synset_id의 word이다. 이 word는 triple의 주어가 된다.

모　듈 :

```
Subject = write.triple.subject(synset_id.word)
```

설　명 : 그리고 다음으로 두 단계의 과정이 이루어 진다. 첫 번째는 찾아진 ptr 값을 OWL의 어휘로 변형한 후, 이를 앞에서 구해진 triple의 서술어로 작성한다.

```
owl_relation = convert.to_owl_vocabulary(ptr)
Predicate = write.triple.predicate(owl_relation)
```

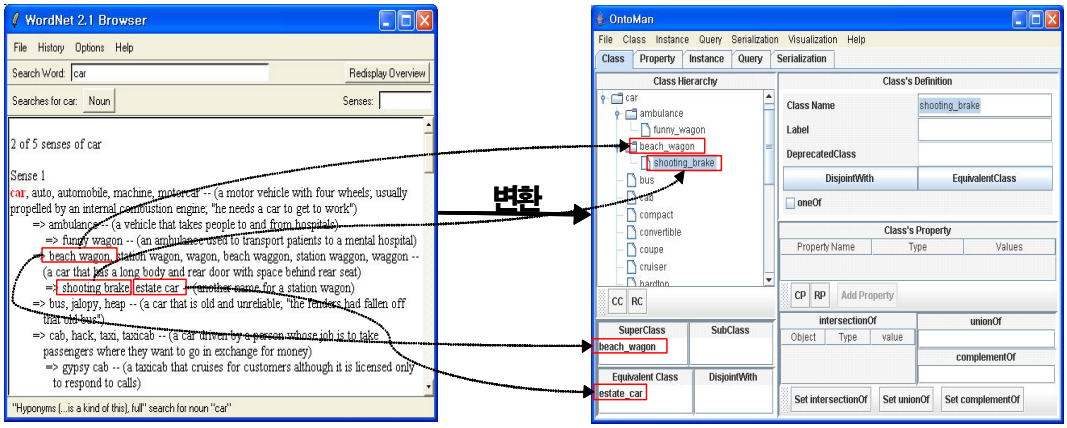
설　명 :끝으로 Step 2에 의해서 얻어진 target_id의 word가 triple의 목적어가 된다.

```
Object = write.triple.object(target_id.word)
```

STEP 4 : 작성된 트리플을 파싱하여 온톨로지로 작성한다.

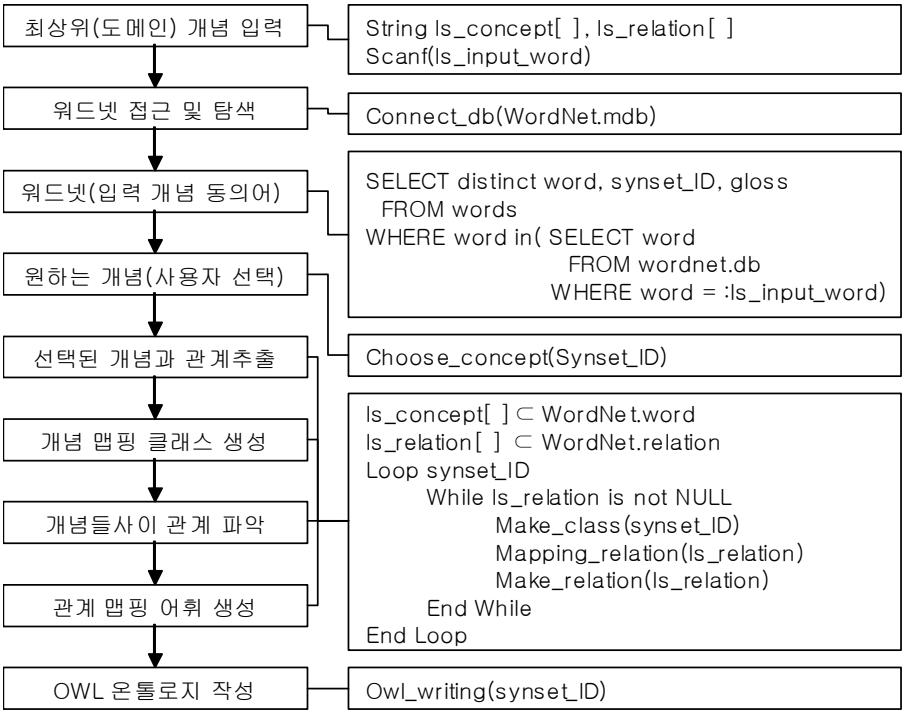
내　용 : 위의 세 단계에서 최종적으로 만들어진 triple set은 파싱과정을 통하여 본 온톨로지 도구에서 온톨로지의 트리 구조로 표현된다.

[그림 9]는 온톨로지 자동 구축 알고리즘을 구현하여 워드넷의 내용을 온톨로지로 자동 구축한 내용이다.



[그림 9] 워드넷 기반 자동 온톨로지 구축 내용

[그림 9]에서와 같이, 온톨로지 자동 구축 알고리즘을 시스템적으로 실현하여 온톨로지 자동 구축 방법을 구현하였다. 온톨로지 자동 구축 모듈의 전체적인 프로세스와 해당 프로세스에 해당하는 중요한 모듈들의 의사코드를 순서적으로 작성해 놓은 것이다.



[그림 10] 온톨로지 자동 구축 프로세스

온톨로지 자동 구축 방법을 평가하기 위해 W3C에서 제공한 와인 온톨로지와 본 연구를 통하여 자동 구축된 와인 온톨로지를 비교 평가한 결과 비슷한 온톨로지가 자동으로 구축되어지는 것을 확인할 수 있었다. 온톨로지 자동 구축 모듈은 온톨로지 개발 방법론에서 제시하는 온톨로지 구축과정에서 클래스 수집, 클래스 간의 관계 정의, 클래스 작성, 클래스 간의 관계 작성 등의 과정을 자동으로 수행한다. 그리하여 사용자는 온톨로지의 구축의 시간을 단축할 수 있으며, 온톨로지 표준화도 기대할 수 있다.

	Wine 온톨로지 (W3C)	Wine 온톨로지 (자동구축)	Food 온톨로지 (W3C)	Food 온톨로지 (자동구축)
해당 온톨로지 일부분				
개념수	74	62	65	2310
속성수	13	4	4	4
인스턴스 수	161	0	45	

[그림 11] 자동 온톨로지 구축 결과 비교

[그림 11]의 와인과 음식에 대한 자동 온톨로지 구축 결과에서 확인할 수 있듯이, W3C에서 제공하는 Wine 온톨로지와 Food 온톨로지의 클래스 및 속성 개수를 비교한 결과 비슷한 정보의 온톨로지가 자동으로 구축됨을 확인 할 수 있다. 특히, Food 온톨로지에서는 기존의 Food 온톨로지에서 포함하고 있는 클래스의 수 보다 상당히 많은 클래스들이 생성되었다. 이는 기존의 온톨로지 구축과정에서 가장 시간이 오래 소요되는 개념들의 수집 및 개념생성 그리고 개념들 간의 관계 설정의 부분을 자동으로 처리하여 온톨로지 구축 시 시간적 단축 효과를 기대할 수 있다. 그러나 온톨로지 내 인스턴스는 반영을 하지 못하는 단점이 있다. 본 결과에서도 알 수 있듯이, 온톨로지 자동 구축 알고리즘을 통하여, 사용자는 클래스의 수집과 클래스의 생성, 클래스들 간의 관계 설정 및 표현의 일련의 과정을 자동으로 수행할 수 있으며, 클래스들이 가지고 있는 속성과 이에 해당되는 인스턴스들의 생성은 사용자에 의해 수동으로 처리되어야할 부분이다.

제 3 절 온톨로지 파싱

시맨틱 웹에서 또 하나의 핵심 연구는 온톨로지의 파싱과 추론이다. 구축된 온톨로지의 효율적 파싱과 추론은 시맨틱 웹의 궁극적인 목적인 의미적인 정보검색의 기반이 된다. 본 연구에서는 OWL Full 문서를 파싱하는 범용적인 파싱 방법을 제안한다. 본 연구에서는 OWL로 작성된 모든 온톨로지 문서를 파싱하는 범용적인 파서와 각각의 OWL 어휘가 갖고 있는 Axiom들을 최대한 반영하여 온톨로지가 기술하고 있는 사실과 암묵적으로 내포하고 있는 정보들을 추론하기 위한 트리플 기반의 파서를 설계하였다.

HP Labs Semantic Web research group에서 연구 중인 Jena[11]는 시맨틱 웹을 위한 어플리케이션들을 위해 작성된 Java 프레임웍이다. Jena는 RDF(S), OWL과 DAML+OIL의 파싱과 추론을 위한 프로그램 환경을 제공한다. Jena는 기존의 추론 엔진들 중 가장 활발히 연구되고 있으며, 시맨틱 웹 실현에 기여도가 상당히 높다. 하지만, Jena는 OWL문서의 파싱과 추론에 대해서는 OWL Lite 어휘 부분만을 다루고 있어 OWL DL과 Full에서 사용되는 어휘들(owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, owl:intersectionOf, owl:hasValue 등)을 분석과 추론이 불가능하다는 한계점이 존재한다. Hoolet[13]은 OWL의 파싱과 처리를 위해 WonderWeb OWL API와 추론을 위해 확장이 용이한 First-Order Translation방법을 기반으로 추론을 하는 OWL DL을 위한 추론엔진이다. 이처럼 간단한 특징을 가진 Hoolet은 Jena처럼 전용추론엔진이라기 보다는 파싱과 일관성 체크엔진 수준의 시스템이다. OWL 추론엔진 JESS[14, 15]는 미국의 Carnegie Mellon 대학에서 개발한 OWL 추론 엔진은 가장 체계적이고 완성도 높은 추론엔진이라 할 수 있다. Jena와 F-OWL[12]등의 추론엔진에서 다루지 못했던 OWL Full의 어휘들까지 모두 추론할 수 있으며, Rule 엔진인 Jess를 기반으로 만들어졌다. JESS는 JAVA 기반의 추론환경을 제공하는 공개소스이며, 정의된 규칙들을 이용하여 새로운 지식 추론과 질의어들을 다룰 수 있는 강력한 엔진이다.

본 연구에서는 효율적 온톨로지 파싱과 파싱된 트리플 추론을 위해 JESS의 추론 모듈을 채택하여 추론 규칙과 내용을 분석한 후, 온톨로지 파싱 방법에 적용하였다.

[표 11] OWL 어휘들의 고유관계

Statement	Predicate	Subject	Object
Resource is a Class	rdf:type	rdfs:Resource	owl:Class
Class is a subtype of Resource	rdfs:subClassOf	owl:Class	rdfs:Resource
Class is a Class	rdf:type	owl:Class	owl:Class
owl:Thing is a Class	rdf:type	owl:Thing	owl:Class
owl:Nothing is a Class	rdf:type	owl:Nothing	owl:Class
Property is a Class	rdf:type	rdf:Property	owl:Class
Domain is a Property	rdf:type	rdfs:domain	rdf:Property
Range is a Property	rdf:type	rdfs:range	rdf:Property
'Data Type Property' is a Class	rdf:type	owl:DatatypeProperty	owl:Class
'Data Type Property' is a subtype of Property	rdfs:subClassOf	owl:DatatypeProperty	rdf:Property
'Object Property' is a Class	rdf:type	owl:ObjectProperty	owl:Class
'Object Property' is a subtype of Property	rdfs:subClassOf	owl:ObjectProperty	rdf:Property
Type is an 'Object Property'	rdf:type	rdf:type	owl:ObjectProperty
'Transitive Property' is a subtype of ObjectProperty	rdfs:subClassOf	owl:TransitiveProperty	owl:ObjectProperty
'Sub Class Of' is a 'Transitive Property'	rdf:type	rdfs:subClassOf	owl:TransitiveProperty
'Sub Property Of' is a 'Transitive Property'	rdf:type	rdfs:subPropertyOf	owl:TransitiveProperty
'Symmetric Property' is a subtype of ObjectProperty	rdfs:subClassOf	owl:SymmetricProperty	owl:ObjectProperty
'Inverse Of' is a Symmetric Property	rdf:type	owl:inverseOf	owl:SymmetricProperty
'Equivalent Property' is a Symmetric Property	rdf:type	owl:equivalentProperty	owl:SymmetricProperty
'Equivalent Class' is a Symmetric Property	rdf:type	owl:equivalentClass	owl:SymmetricProperty
'Same Individual' is a Symmetric Property	rdf:type	owl:sameIndividualAs	owl:SymmetricProperty
'Complement Of' is a Symmetric Property	rdf:type	owl:complementOf	owl:SymmetricProperty
'Different From' is a Symmetric Property	rdf:type	owl:differentFrom	owl:SymmetricProperty
'Disjoint With' is a Symmetric Property	rdf:type	owl:disjointWith	owl:SymmetricProperty
'Equivalent Property' is a Transitive Property	rdf:type	owl:equivalentProperty	owl:TransitiveProperty
'Equivalent Class' is a Transitive Property	rdf:type	owl:equivalentClass	owl:TransitiveProperty
'Same Individual' is a Transitive Property	rdf:type	owl:sameIndividualAs	owl:TransitiveProperty
'Functional Property' is a subtype of Property	rdfs:subClassOf	owl:FunctionalProperty	rdf:Property
'Inverse Functional Property' is a subtype of Property	rdfs:subClassOf	owl:InverseFunctionalProperty	rdf:Property
sameAs is equivalent to sameIndividualAs	owl:sameAs	owl:equivalentProperty	owl:sameIndividualAs

JESS는 OWL 어휘들이 갖고 있는 고유관계들 30가지와 각 어휘에 적용될 추론 규칙 20가지를 정의하고 있다. 어휘들 사이의 고유관계는 [표 11]과 같이 정의하였으며, 어휘들 간의 관계와 어휘가 갖고 있는 고유 Axiom들을 이용하여 [표

12]의 20가지 추론 규칙을 정의하고 있다.

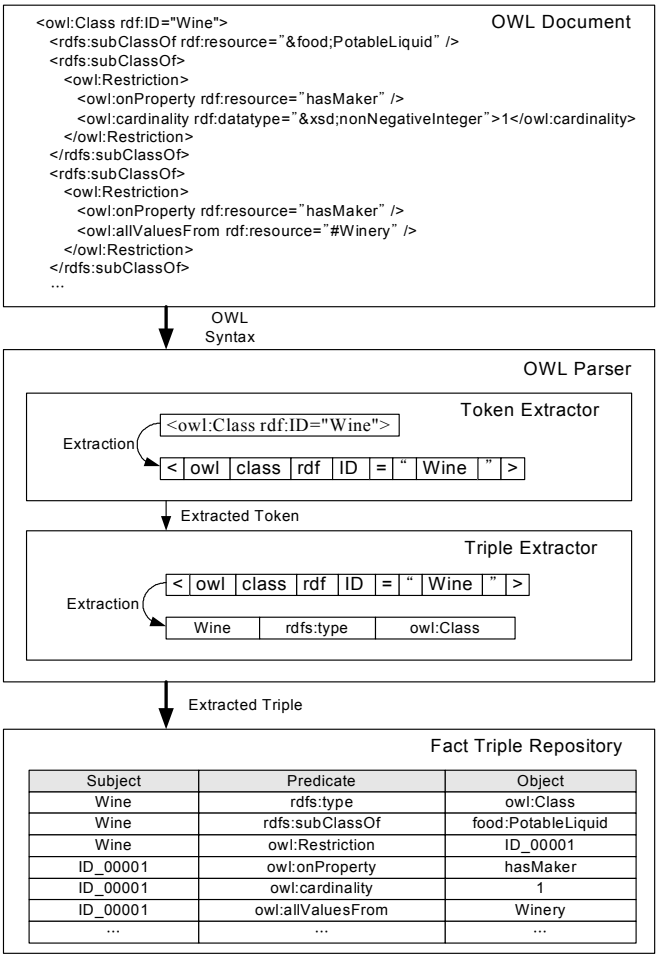
[표 12] JESS의 추론규칙

규칙 이름		추론 방식
1	Propagate Transitive Properties	Triple(S ?p)(P rdf:type)(O owl:TransitiveProperty) Triple(S ?x)(P ?p)(O ?y) Triple(S ?y)(P ?p)(O ?z) => Assert Triple(S ?x)(P ?p)(O ?z)
2	Complete Typing of instances of objects with inherited types	Triple(S ?y)(P rdfs:subClassOf)(O ?z) Triple(S ?x)(P rdf:type)(O ?y) => Assert Triple(S ?x)(P rdf:type)(O ?z)
3	Complete typing of instances of properties with inherited types	Triple(S ?y)(P rdfs:subPropertyOf)(O ?z) Triple(S ?a)(P ?y)(O ?b) => Assert Triple(S ?a)(P ?z)(O ?b)
4	Default behavior for signature propagation	Triple(S ?x)(P rdfs:subPropertyOf)(O ?y) Triple(S ?y)(P rdfs:domain)(O ?z) => Assert Triple(S ?x)(P rdfs:domain)(O ?z)
		Triple(S ?x)(P rdfs:subPropertyOf)(O ?y) Triple(S ?y)(P rdfs:range)(O ?z) => Assert Triple(S ?x)(P rdfs:range)(O ?z)
5	Complement with instances of symmetric properties	Triple(S ?p)(P rdf:type)(O owl:SymmetricProperty) Triple(S ?x)(P ?p)(O ?y) => Assert Triple(S ?y)(P ?p)(O ?x)
6	Complete with instances of inverse properties	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?x)(P ?p1)(O ?y) => Assert Triple(S ?y)(P ?p2)(O ?x)
7	The inverse property of a functional property is an inverse functional property	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?p1)(P rdf:type)(O owl:FunctionalProperty) => Assert Triple(S ?p2)(P rdf:type)(O owl:inverseFunctionalProperty)
8	The inverse property of an inverse functional property is a functional property	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?p1)(P rdf:type)(O owl:inverseFunctionalProperty) => Assert Triple(S ?p2)(P rdf:type)(O owl:FunctionalProperty)
9	Complete with instances of equivalent Class	Triple(S ?t1)(P owl:equivalentClass)(O ?t2) Triple(S ?x)(P rdf:type)(O ?t1) => Assert Triple(S ?x)(P rdf:type)(O ?t2)
10	Complete hierarchy with equivalent classes: if t1<=>t2 then t1<t2, and by symmetry we also get t2<t1	Triple(S ?t1)(P owl:equivalentClass)(O ?t2) => Assert Triple(S ?t1)(P rdfs:subClassOf)(O ?t2) => Assert Triple(S ?t2)(P rdfs:subClassOf)(O ?t1)
11	Members of disjoint classes are different instances	Triple(S ?c1)(P owl:disjointWith)(O ?c2) Triple(S ?o1)(P rdf:type)(O ?c1) Triple(S ?o2)(P rdf:type)(O ?c2) => Assert Triple(S ?o1)(P owl:differentFrom)(O ?o2)
12	Two classes subsuming each other are equivalent	Triple(S ?c1)(P rdfs:subClassOf)(O ?c2) Triple(S ?c2)(P rdfs:subClassOf)(O ?c1) Test(nnq ?c1 ?c2) => Assert Triple(S ?c1)(P owl:equivalentClass)(O ?c2)

13	Complete instances with properties of equivalent instances	Triple(S ?s1)(P owl:sameIndividualAs)(O ?s2) Triple(S ?s1)(P ?p)(O ?o) => Assert Triple(S ?s2)(P ?p)(O ?o)
14	Complete instances of properties with instances of equivalent properties	Triple(S ?p1)(P owl:equivalentProperty)(O ?p2) Triple(S ?s)(P ?p1)(O ?o) => Assert Triple(S ?s)(P ?p2)(O ?o)
15	Complete hierarchy with equivalent properties: if p1<=>p2 then p1<p2, and by symmetry we also get p2<p1	Triple(S ?p1)(O owl:equivalentProperty)(O ?p2) => Assert Triple(S ?p1)(P rdfs:subPropertyOf)(O ?p2) => Assert Triple(S ?p2)(P rdfs:subPropertyOf)(O ?p1)
16	Two properties subsuming each other are equivalent	Triple(S ?p1)(P rdfs:subPropertyOf)(O ?p2) Triple(S ?p2)(P rdfs:subPropertyOf)(O ?p1) Test(neq ?p1 ?p2) => Assert Triple(S ?p1)(P owl:equivalentProperty)(O ?p2)
17	Derive equivalence form multiple instantiations of a functional property with the same subject	Triple(S ?p)(P rdf:type)(O owl:FunctionalProperty) Triple(S ?s)(P ?p)(O ?o1) Triple(S ?s)(P ?p)(O ?o2) Test(neq ?o1 ?o2) => Assert Triple(S ?o1)(P owl:sameIndividualAs)(O ?o2)
18	Derive equivalence from multiple instantiations of inverse functional property with the same object	Triple(S ?p)(P rdf:type)(O owl:InverseFunctionalProperty) Triple(S ?s1)(P ?p)(O ?o) Triple(S ?s2)(P ?p)(O ?o) Test(neq ?s1 ?s2) => Assert Triple(S ?s1)(P owl:sameIndividualAs)(O ?s2)
19	Complete the extension of the Thing class with all other classes	Triple(S ?s)(P rdf:type)(O ?o) => Assert Triple(S ?s)(P rdf:type)(O owl:Thing)
20	A property inverse of it-self is symmetric	Triple(S ?p)(P owl:inverseOf)(O ?p) Assert Triple(S ?p)(P rdf:type)(O owl:SymmetricProperty)

본 연구에서의 추론 방법은 JESS에서 정의하고 있는 어휘들의 관계정의와 추론 규칙을 기반으로 OWL에서 추론가능한 모든 어휘를 처리할 수 있도록 설계하였다. 트리플을 처리하는 것은 OWL 문서의 모든 구문을 파싱하여 구조화된 형태인 Statement(주어-서술어-목적어)로 변환하고, 이를 이용하여 추론 규칙을 적용한 2차 파싱을 수행하는 일련의 과정이다. 본 연구의 온톨로지 파싱 방법은 [표 11]에 정의된 30가지 OWL의 구조적 규칙에 의한 온톨로지가 포함하고 있는 Fact 트리플을 추출하는 1차 파싱과 이렇게 파싱된 Fact 트리플에서 [표 12]에 정의된 20가지 추론 규칙을 적용한 2차 파싱을 의미한다. 본 연구에서 온톨로지가 기술하는 모든 사실을 빠르게 추출하기 위해 2단계에 걸친 온톨로지 파서를 설계하였다. 정확하고 빠른 파서의 설계를 위해 토큰 추출기에서 온톨로지의 구문의 토큰을 1단계로 추출하고, 이를 바탕으로 트리플 추출기에서 Statement를 추출한

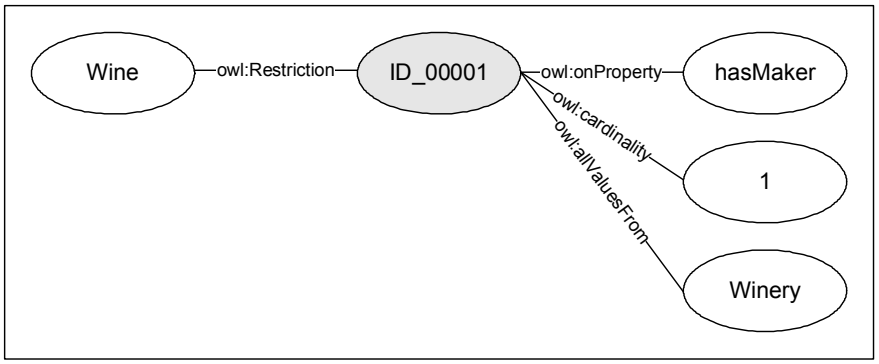
다. 본 연구에서 제안하는 추론 규칙 기반의 OWL 온톨로지 파서 설계도는 [그림 12]와 같다.



[그림 12] 파서 처리 과정

온톨로지 구문의 파싱을 위해 작성된 OWL 온톨로지를 입력한다. 온톨로지 파서 내의 토근 추출기에서 '<'에서 '>'까지의 구문으로부터 각각의 토큰을 추출해 낸다. 그리고 추출된 토큰을 트리플 추출기로 전달을 하여, 온톨로지에 정의된 어휘와 구문에 작성된 자원을 이용하여 온톨로지가 기술하고 있는 Fact를 각각의 트리플들로 추출한다. 마지막으로 추출된 트리플은 Fact 트리플 저장소에 저장된

다. OWL은 사용의 목적에 따라 Lite, DL과 Full의 세 가지로 분류되는데, 모든 OWL 구문의 원활한 파싱을 위해서는 Blank Node를 다루어야만 완벽한 파싱이 가능하다. OWL 구문은 단순 트리플만으로는 1차원의 파싱(자원과 자원 사이의 관계 정의)만이 가능하여, 풍부한 OWL 내용을 모두 표현하는데 제한적이다. 특히 enumerated datatype(owl:oneOf), local restriction Property(owl:Restriction), individual axioms(owl:Alldifferent), boolean class expressions (owl:intersectionOf, owl:complementOf, owl:unionOf) 등의 자원 정의 부분은 반드시 Blank Node를 사용하여 파싱해야 한다. 위의 [그림 13]에서 클래스 'Wine'을 정의할 때, 지역 제약 속성으로 'hasMaker'를 갖고, 이 속성의 값은 'owl:cardinality'를 이용하여 반드시 '1'개의 값을 가질 수 있다고 정의하고 있다. 또한 'hasMaker'의 값으로는 'owl:allValuesFrom'을 이용하여 클래스 'Winery'의 인스턴스들로 정의됨을 기술하고 있다. 본 연구에서는 [그림 13]과 같이 특정의 ID를 이용하여 Blank Node로 처리하여 속성이 갖는 지역 제약의 내용을 모두 기술하도록 설계하였다.



[그림 13] Blank Node 처리

온톨로지에 작성된 모든 구문을 파싱하여 Fact 트리플을 추출하고, 만들어진 트리플을 Fact 트리플 저장소에 저장함으로써 1차 파싱은 완료된다. 그리고 이렇게 추출된 트리플들은 추론과 질의처리 모듈의 기반이 된다. 본 연구의 파싱에서는 단순 구문에 대한 파싱과 더불어 내부적 의미에 대한 파싱을 동시에 수행한다. 1차 파싱을 통해 추출된 구문적 Fact 트리플과 OWL이 정의하고 있는 어휘들의

Axiom들이 의미하고 있는 Semantic 트리플을 이용하여 의미적인 정보검색을 기대한다. 새로운 Semantic 트리플을 만들기 위해 OWL에 정의된 각 어휘의 Axiom이 포함하고 있는 추론 규칙들을 정의하였다. 추론 규칙은 전이추론, 인스턴스 역-전이추론, 속성 상속 추론, domain 및 range 상속 추론, 대칭추론 등 모두 20가지이며, [표 13]은 대표적으로 전이, 대칭, domain 및 range 상속과 속성 상속 추론에 대한 추론 규칙과 적용되는 OWL 어휘를 보이고 있다.

[표 13] 대표적 추론 규칙과 적용되는 OWL 어휘들

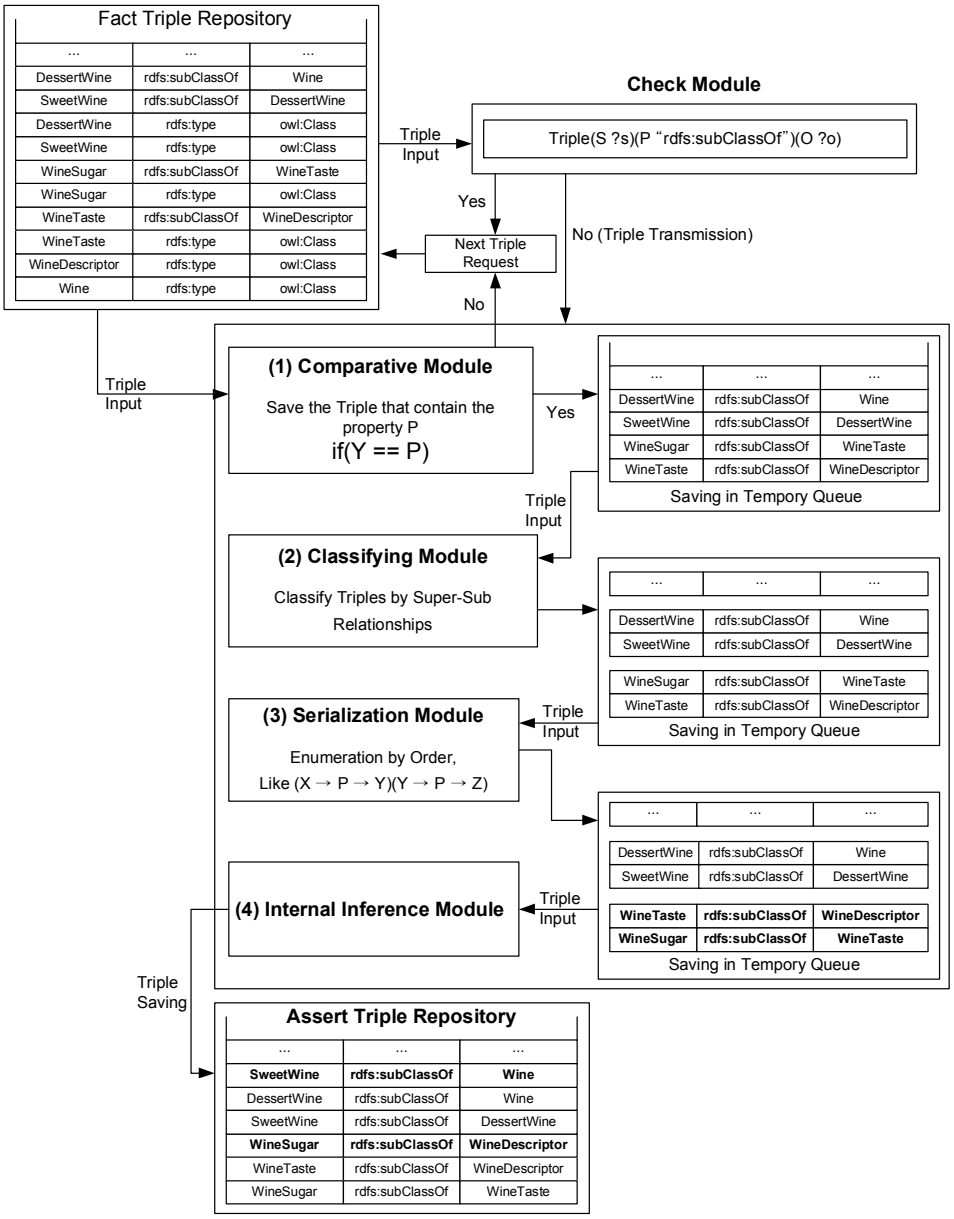
추론	규칙	적용어휘
전이	Triple(Subject ?p)(Predicate "http://-#type")(Object "http://-#TransitiveProperty") Triple(Subject ?x)(Predicate ?p)(Object ?y), Triple(Subject ?y)(Predicate ?p)(Object ?z) =>Assert(Triple(Subject ?x)(Predicate ?p)(Object ?z))	equivalentClass, subClassOf,sameAs, TransitiveProperty, equivalentProperty, subPropertyOf
대칭	Triple(Subject ?p)(Predicate "http://-#type")(Object "http://-#symmetricProperty") Triple(Subject ?x)(Predicate ?p)(Object ?y) =>Assert(Triple(Subject ?y)(Predicate ?p)(Object ?x))	sameAs, inverseOf, equivalentProperty, SymmetricProperty,equivalentClass, differentFrom, disjointWith, complementOf
속성 상속 추론	Triple(Subject ?p1)(Predicate "http://-#inverseOf")(Object ?p2) Triple(Subject ?x)(Predicate ?p1)(Object ?y) =>Assert(Triple(Subject ?y)(Predicate ?p2)(Object ?x))	inverseOf, inverseFunctionalProperty, FunctionalProperty
범주 속성 상속	Triple(Subject ?x)(Predicate "http://-#subPropertyOf")(Object ?y) Triple(Subject ?y)(Predicate "http://-#domain")(Object ?z) =>Assert(Triple(Subject ?x)(Predicate "http://-#domain")(Object ?z))	range, domain

본 연구에서는 이러한 추론 규칙을 효율적으로 파싱하여 Semantic 트리플을 만들기 위해 추론 모듈은 체크모듈, 비교모듈, 분류모듈, 순서화모듈 그리고 내부추론모듈의 다섯 가지 모듈을 설계하여, 각각의 규칙들에 맞게 설계된 내부 추론 모듈들을 적용하여 OWL 온톨로지에서 Semantic 트리플들을 파싱하였다. 각 추론 규칙에 해당하는 모듈들에 대한 설명은 [표 14]와 같다.

[표 14] 추론 규칙에 대한 트리플 변환 모듈

모듈	전이추론	대칭추론	domain, range 상속추론	속성상속추론
체크 모듈	전이되는 특징을 갖는 어휘인지 파악하고, 이미 추론되었는지 체크	대칭속성 또는 대칭속성의 인스턴스인지 파악하고, 이미 추론되었는지 체크	특정 속성이 domain 또는 range를 갖는다면, 그 속성의 하위속성이 있는지 파악하고, 이미 추론되었는지 체크	특정 클래스가 속성을 가질 때, 그 클래스의 하위 클래스가 있는지 파악하고, 이미 추론되었는지 체크
비교 모듈	입력받은 Fact 트리플이 추론하고자 하는 트리플인지 비교하고, 맞다면 임시큐에 저장			
분류 모듈	임시 큐에 저장된 정렬되지 않은 트리플들을 연속된 관계에 있는 트리플로 분류			
순서화 모듈	연속된 관계에 있는 트리플들을 최하위부터 최상위까지 일관된 순서로 열거		임시 큐에 있는 트리플들을 domain 또는 range를 갖고 있는 속성부터 최하위 속성까지 일관된 순서로 열거	임시 큐에 있는 트리플들을, 속성을 갖고 있는 클래스부터 최하위 클래스까지 일관된 순서로 열거
내부 추론 모듈	순서대로 나열된 트리플들을 '배분법칙'에 의하여 새로운 트리플 추출	각 트리플 (S-P-O)을 대칭 Axiom에 따라, 새로운 트리플 (O-P-S)을 생성함	순서대로 나열된 속성들에 domain과 range를 부여함	순서대로 나열된 클래스들에 속성들을 부여함

[표 13]의 추론 규칙들은 각각 본 연구에서 설계한 다섯 가지의 변환 모듈들을 [표 14]와 같이 적용하여 파싱을 수행하고, 본 연구의 파서에 의해 1차 파싱을 통해 만들어진 Fact 트리플들을 하나씩 체크함으로써 2차 파싱이 시작된다. [그림 14]는 전이추론 중 'rdfs:subClassOf'어휘에 대한 추론방법을 나타내고 있다. [그림 14]에서 알 수 있듯이 전이 추론에서는 트리플 생성을 위해, 본 연구에서 제안하고 있는 체크모듈, 비교모듈, 분류모듈, 순서화모듈, 내부추론모듈을 모두 거쳐서 Semantic 트리플 파싱 작업이 이루어지고 있으며, [표 14]의 대칭추론은 이러한 추론 모듈 중 체크모듈과 비교모듈 그리고 내부추론 모듈만으로도 Semantic 트리플 파싱이 가능하다. 부록 1에 2차 파싱을 위한 각각의 추론 규칙들과 규칙들에 대한 파싱 모듈이 있다.



[그림 14] 전이-'rdfs:subClassOf'

[그림 14]는 추론 규칙 중 전이추론에 대한 2차 파싱 방법에 대한 내용이다. [그림 14]에서도 알 수 있듯이 전이추론에 대한 2차 Semantic 트리플 파싱에서는 다섯 가지의 추론 모듈이 적용되어 파싱이 이루어지고 있다.

(1) Fact 트리플 저장소에 있는 Fact 트리플을 하나씩 입력받는다.

(2) 체크모듈 : Predicate가 'rdfs:subClassOf'일 경우, 이 어휘에 대한 추론이 이미 되었는지 여부를 체크하여, 추론되지 않은 트리플일 경우 제 1의 임시 큐에 저장한다.

(3) 비교모듈 : Fact 트리플 저장소에 있는 모든 트리플을 입력으로 받아, Predicate가 'rdfs:subClassOf'인 트리플만을 제1의 임시 큐에 축적한다.

(4) 분류모듈 : 임시 큐에 저장된 정렬되지 않은 트리플들을 상-하위 관계에 있는 트리플들로 분류하여 제2의 임시 큐에 저장한다.

(5) 순서화모듈 : 상-하위 관계로 정의된 트리플들을 최하위 클래스부터 최상위 클래스로의 일정한 순서로 열거하여 제3의 임시 큐에 저장한다.

(6) 내부추론모듈 : 일정한 순서로 나열된 트리플들을 '배분법칙'에 의하여 새로운 트리플을 추출하고, Assert 트리플 저장소에 추론된 Semantic 트리플을 저장한다.

[그림 14]에서 체크모듈과 비교모듈에서 'P'의 대상을 다른 전이추론의 어휘인 'owl:equivalentClass', 'owl:sameAs', 'owl:equivalentProperty', 'rdfs:subPropertyOf'로 바꾸면, 같은 방식으로 전이추론 결과를 얻을 수 있다. 이와 같은 방법으로 모든 추론 규칙에 해당하는 OWL 어휘들에 대한 2차 파싱이 이루어진다. 20가지 추론규칙과 각각의 추론 규칙에 대한 내부 추론 모듈을 이용한 2차 파싱을 거쳐 의미적 트리플이 만들어진다.

[그림 15]와 [그림 16]은 각각 OWL 데이터에서 클래스 부분에 해당하는 파싱 과정과 파싱 결과와 속성 부분에 해당하는 파싱 과정과 결과를 보여주고 있다.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03owlt/Ontology/premises001#"
  xml:base="http://www.w3.org/2002/03owlt/Ontology/premises001" >

  <owl:Ontology rdf:about="" />

  <owl:Class rdf:ID="Car">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Automobile"/>
    </owl:equivalentClass>
  </owl:Class>
  <first:Car rdf:ID="car">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Car>
  <first:Automobile rdf:ID="auto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Automobile>
</rdf:RDF>
```

주어	서술어	목적어
first:Car	rdf:type	owl:Class
first:Automobile	rdf:type	owl:Class
first:Car	owl:equivalentClass	first:Automobile
first:car	rdf:type	first:Car
first:car	rdf:type	owl:Thing
first:auto	rdf:type	first:Automobile
first:auto	rdf:type	owl:Thing

[그림 15] OWL 1차 Syntax 파싱 예

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03owlt/equivalentProperty/premises001#"
  xml:base="http://www.w3.org/2002/03owlt/equivalentProperty/premises001" >

  <owl:ObjectProperty rdf:ID="hasHead">
    <owl:equivalentProperty>
      <owl:ObjectProperty rdf:ID="hasLeader"/>
    </owl:equivalentProperty>
  </owl:ObjectProperty>
</rdf:RDF>
```

주어	서술어	목적어
first:hasHead	rdf:type	owl:ObjectProperty
first:hasLeader	rdf:type	owl:ObjectProperty
first:hasHead	owl:equivalentProperty	first:hasLeader

[그림 16] OWL 2차 Semantic 파싱 예

OWL 구문의 1차 파싱을 거친 Fact 트리플과 2차 파싱 모듈에 의해 새롭게 재구성된 Semantic 트리플들을 모두 이용하여 의미적 정보검색을 실현할 수 있다. 본 연구를 통해 설계한 온톨로지 파서 이용하여 와인 온톨로지에 대해 파싱을 수행하였다. 본 시스템의 파서를 통해 [그림 17]과 같이 총 913개의 Fact 트리플을 얻을 수 있었다. 그러나 1차 파싱을 통한 파싱된 결과를 이용하여 기본적인 트리플 구조에 기반한 정보 검색은 가능하나 그 검색 결과가 의미적 정보를 제공한다고 할 수 없다. 그리하여, 2차 파싱을 통하여 Fact 트리플들에 대한 20가지의 추론규칙을 통해 Semantic 사실들을 파싱한 결과 [그림 18]과 같이 847개의 새로운 Semantic 트리플들을 추출할 수 있었다. 본 시스템의 파싱 방법은

‘owl:hasValue’, ‘owl:disjointWith’ 등과 같이 OWL DL과 OWL Full 어휘까지도 추론규칙에 적용하여 파싱이 가능하기 때문에 시맨틱 웹에서의 의미적 정보검색에 활용 가능성이 높다.

19	WhiteWine	owl:intersectionOf	hasColor
20	WhiteWine	hasColor	White
21	WhiteTableWine	rdf:type	owl:Class
22	WhiteTableWine	owl:intersectionOf	TableWine
23	WhiteTableWine	owl:intersectionOf	hasColor
24	WhiteTableWine	hasColor	White
25	WhiteNonSweetWine	rdf:type	owl:Class
26	WhiteNonSweetWine	owl:intersectionOf	WhiteWine
27	WhiteNonSweetWine	owl:intersectionOf	hasSugar
28	WhiteNonSweetWine	hasSugar	Dry
29	WhiteNonSweetWine	hasSugar	OffDry
30	WhiteLoire	rdf:type	owl:Class
31	WhiteLoire	owl:intersectionOf	Loire
32	WhiteLoire	owl:intersectionOf	WhiteWine
33	WhiteLoire	owl:Restriction	madeFromGrape
34	WhiteBurgundy	rdf:type	owl:Class
35	WhiteBurgundy	owl:intersectionOf	Burgundy
36	WhiteBurgundy	owl:intersectionOf	WhiteWine
37	WhiteBurgundy	madeFromGrape	ChardonnayGrape
38	WhiteBurgundy	owl:Restriction	madeFromGrape
39	WhiteBurgundy	owl:Restriction	madeFromGrape
40	WhiteBordeaux	rdf:type	owl:Class
41	WhiteBordeaux	owl:intersectionOf	Bordeaux
42	WhiteBordeaux	owl:intersectionOf	WhiteWine
43	WhiteBordeaux	owl:Restriction	madeFromGrape
44	Region	rdf:type	owl:Class
45	locatedIn	rdf:type	owl:TransitiveProperty

[그림 17] OWL 어휘기반 Fact 트리플 생성

79	LateHarvest	rdfs:subClassOf	Wine
80	LateHarvest	rdfs:subClassOf	food;PotableLiquid
81	Wine	rdfs:subClassOf	food;PotableLiquid
82	WineGrape	rdfs:subClassOf	food;Grape
83	WineSugar	rdfs:subClassOf	WineTaste
84	WineSugar	rdfs:subClassOf	WineDescriptor
85	WineTaste	rdfs:subClassOf	WineDescriptor
86	WineColor	rdfs:subClassOf	WineDescriptor
87	WineFlavor	rdfs:subClassOf	WineTaste
88	WineBody	rdfs:subClassOf	WineTaste
89	SweetRiesling	rdfs:subClassOf	DessertWine
90	SweetRiesling	rdfs:subClassOf	Wine
91	DessertWine	rdfs:subClassOf	Wine
92	Sauternes	rdfs:subClassOf	Bordeaux
93	Port	rdfs:subClassOf	RedWine
94	EarlyHarvest	rdfs:subClassOf	Wine
95	Sauternes	rdfs:subClassOf	LateHarvest
96	Chianti	rdfs:subClassOf	ItalianWine
97	Tours	locatedIn	ToursRegion
98	Tours	locatedIn	LoireRegion
99	Tours	locatedIn	FrenchRegion
100	ToursRegion	locatedIn	LoireRegion
101	ToursRegion	locatedIn	FrenchRegion
102	LoireRegion	locatedIn	FrenchRegion
103	Sancerre	locatedIn	SancerreRegion
104	Sancerre	locatedIn	LoireRegion
105	SancerreRegion	locatedIn	LoireRegion

[그림 18] 추론 규칙 적용 Semantic 트리플 생성

추론 모듈을 통하여 추론된 트리플들은 트리플 기반 질의 처리 방법의 입력 데이터로 사용된다.

제 4 절 온톨로지 질의 처리

온톨로지를 응용하기 위해 온톨로지 내 효율적 정보 탐색이 가능하여야 한다. 온톨로지내의 효율적 정보 탐색을 위한 질의 처리 부분은 온톨로지 응용의 측면에서 반드시 해결되어야 할 부분이다. 기존의 온톨로지에 대한 질의 처리 시스템은 온톨로지에 표현된 사실만을 분석하여, 단순 구조적 사실 정보만을 사용자에게 제공함으로써 데이터베이스 시스템 및 텍스트 기반 정보처리 시스템과 크게 다른 점이 없었다. 사실상, 기존의 온톨로지 질의 시스템들에서 사용하고 있는 질의 언어의 구조나 형식이 데이터베이스 시스템의 질의 언어(SQL)에 모태를 두고 있으므로, 온톨로지와 데이터베이스의 질의 처리 및 그 결과는 거의 동일하다. 이에 본 연구에서는 온톨로지 사용의 효율성을 극대화하기 위해 온톨로지에 대한 단순 질의 처리가 아닌, 추론 규칙에 기반한 추론된 사실들을 모두 처리할 수 있는 온톨로지 추론 및 질의에 대한 통합 시스템 개발의 필요성을 인식하고, 온톨로지에 대한 효율적 질의 처리 방법을 연구하였다.

본 시스템에서는 기존 온톨로지 질의 언어(RQL, RDQL, SPARQL)들이 가지고 있는 질의 구문 작성 시 복잡성을 해결하기 위해 유도 구문 기반의 질의 처리가 가능한 질의 처리 인터페이스를 설계하였다. 사용자는 텍스트 입력 부분에 온톨로지 내 유도 질의를 작성하여 온톨로지에 대한 질의 처리를 수행한다. 또 2차 파싱을 거쳐 생성된 추론된 트리플들에 기반한 온톨로지 질의 처리가 수행된다. 끝으로, 질의 처리의 용이성과 트리플 사실 기반의 추론을 통하여 사용자는 온톨로지 구축의 완전성 및 응용 가능성을 쉽게 평가할 수 있는 특징이 있다. 기존의 온톨로지 저작도구에서는 온톨로지 구축 시 온톨로지의 구축이 잘 되었는지 평가할 수 있는 방법이 없었다. 본 모듈은 온톨로지의 저작도구에 추가되어 사용자가 구축한 온톨로지에 대하여 원하는 결과를 설정한 후 이에 맞게 온톨로지를 구축해 나갈 수 있다는 장점이 있다. 그리하여 시맨틱 웹의 핵심이 되는 온톨로지를 정확하고 사용자가 원하는 대로 구축할 수가 있으므로 사용자는 완벽한 정보를 제공할 수 있다. 온톨로지 질의를 위해서 많은 연구가 진행되어져 왔으며, RQL, RDQL, SPARQL과 같은 질의 언어들이 개발되었다. 이러한 언어들의 구조는 데이터베이스 질의 언어인 SQL에 모태를 두고 있다. RQL, RDQL, SPARQL은

RDF 그래프 모델에 기반하여 사용자 질의를 처리함으로써, OWL 파일을 처리하는데 한계를 가지고 있으며, 질의를 수행하기 위해 복잡한 질의 구조를 사용하여 질의를 작성해야 하는 어려움이 있다. 그리고 SQL은 데이터베이스의 일반화로 질의 언어의 사용이 보편화 및 구조화되어 있지만, 역시 데이터베이스를 처리하는데 적합한 언어로써 온톨로지를 처리하는 데는 한계가 있다. 최근의 연구에서는 온톨로지를 데이터베이스화 함으로써 데이터베이스로 변형된 내용에 대하여 SQL을 사용하여 질의를 처리하는 연구도 있었다. 대표적인 데이터베이스 기반 온톨로지는 워드넷이며, 이를 처리하기 위해서 SQL 구문이 사용된다. [표 15]는 기존 온톨로지 질의 언어와 제안하고 있는 질의 처리 방식의 특징을 비교하여 보여주고 있다.

[표 15] 온톨로지 질의 언어 비교

	RQL, RDQL, SPARQL	SQL	제안한 질의 모듈
처리영역	RDF	Database	RDF, OWL
추론기능	없음	없음	있음
기본구조	SELECT, FROM, WHERE		
특 징	RDF 그래프 모델에 기반한 텍스트 매칭의 온톨로지내의 간단한 구조적 정보 검색	온톨로지 검색을 위해서는 온톨로지 파일을 데이터베이스화하는 사전 작업이 요구됨	사용자중심의 텍스트기반 질의 인터페이스 및 추론과 질의의 통합시스템 제공

[표 16]은 본 연구의 질의 처리 방법의 특징을 설명하고 있다.

[표 16] 질의 처리 모듈의 특징

질의 처리 모듈의 특징
● 전문가 : SELECT, FROM, WHERE 질의 지원 ● 일반 사용자 : 자연어 질의 사용 가능 (텍스트 기반 방식) ● 추론된 트리플 파일을 입력 형식으로 사용하여 추론 기능 있음 ● RDF(S)와 OWL 온톨로지 모두 질의 처리 가능

본 연구에서는 기존의 Select, From, Where 질의 구조를 트리플 입력에 맞게 변형하여 본 연구의 질의 처리 모듈에 적용하였다. 본 시스템에서 사용되는 질의 처리 언어는 다음과 같은 형식으로 구성되어져 있다.

STEP 1 : Subject를 추출한다.

내 용 : 추론된 트리플에 기반한 질의 언어에서 우선 몇 가지의 OWL 어휘의 의미를 이해하고 있어야 한다. 이중에 가장 중요한 한 가지가 바로 rdf:type 이다. 본 연구의 질의 처리 모듈에 처음 입력되는 값에 해당되는 모든 내용들은 바로 rdf:type으로 연결된 값이다.

모 듈 :

```
select {SUBJECT}
 from {TRIPLE_SET}
where {rdf:type=:ls_arg1}
```

설 명 : 여기에서는 추론 규칙에 의해서 만들어진 트리플 집합에서 입력된 값과 rdf:type의 관계를 이루고 있는 모든 값들이 바로 사용자가 원하는 Subject가 될 수 있다. 본 연구에서는 미리 추론과정을 거쳐서 만들어진 추론된 트리플 집합을 사용함으로써 위의 모듈에서 더 풍부한 값들을 기대할 수 있지만 일반적인 추론되지 않은 구조적 트리플에서도 위의 모듈을 통해서 원하는 Subject를 추출할 수 있다.

예 제 :

```
select {SUBJECT}
 from Wine.triple
where rdf:type=:wine
```

설 명 : wine과 rdf:type의 관계를 이루고 있는 트리플내의 모든 Subject를 찾는 질의이다. 본 질의의 결과는 아래와 같으며, 결과값은 STEP 2의 입력으로 사용된다.

결 과 :

```
ChateauChevalBlancStEmilion,ChateauDYchemSauterne,ChateauLafiteRothschildPauillac,ChateauMorgonBeaujolais,CorbansPrivateBinSauvignonBlanc,ElyseZinfandel,FormanCabernetSauvignon,GaryFarrellMerlot,StGenevieveTexasWhite,WhitehallLaneCabernetFranc,VentanaCheninBlanc,StonleighSavignonBlanc,SelaksSauvignonBlanc,SelaksIceWine,SeanThackreySiriusPetiteSyrah,SchlossVolradTrochenbierenausleseRiesling,SchlossRothermelTrochenbierenausleseRiesling,SaucelitoCanyonZinfandel1998,SaucelitoCanyonZinfandel,StGenevieveTexasWhite,SantaCruzMountainVineyardCabernetSauvignon,PulignyMontrachetWhiteBurgundy,PeterMccoyChardonnay,PageMillWineryCabernetSauvignon,MountadamRiesling,MountadamPinotNoir,MountadamChardonnay,MountEdenVineyardEstatePinotNoir,MariettaZinfandel,MariettaPetiteSyrah,MariettaOldVinesRed,MariettaCabernetSauvignon,LongridgeMerlot,LaneTannerPinotNoir,KathrynKennedyLateral,KalinCellarsSemillon,GaryFarrellMerlot,FoxenCheninBlanc,FormanChardonnay,FormanCabernetSauvignon,ElyseZinfandel,CotturiZinfandel,CortonMontrachetWhiteBurgundy,CorbansSauvignonBlanc,CorbansPrivateBinSauvignonBlanc,CorbansDryWhiteRiesling,CongressSpringsSemillon,ChiantiClassico,ChateauMargaux,ChateauLafiteRothschildPauillac,ChateauDeMeursaultMeursault,ChateauDYchemSauterne,ChateauChevalBlancStEmilion
```

STEP 2 : STEP 1에서 추출된 개념들을 Subject로 한 트리플 집합내의 모든 Sub 트리플들을 추출한다.

내 용 : STEP 3을 진행하기 위하여 새로운 트리플 집합을 STEP 1의 결과값을 기반으로 구성한다. 이렇게 새롭게 트리플을 구성함으로써 질의 처리 속도를 줄일 수 있다. STEP 2를 거쳐 새로운 트리플 집합_sub1이 만들어진다.

STEP 3 : 트리플 집합_sub1에서 입력된 Predicate와 관련된 모든 트리플들을 추출하여 트리플 집합_sub2를 만든다.

내 용 : STEP 3에서 arg2는 반드시 predicate 정보가 입력되므로 본 연구의 질의처리 모듈에서 트리플내의 질의 처리는 predicate를 조사하여 새로운 트리플 집합_sub2를 만든다.

모 듈 :

```
select {TRIPLE}
 from {TRIPLE_SET_sub1}
where {TRIPLE_SET_sub1.predicate=:ls_arg2}
```

설 명 : 여기에서, 다시 트리플 집합_sub2가 만들어지며, 이렇게 만들어진 트리플 집합_sub2는 STEP 4의 입력이 된다.

예 제 :

```
select {PREDICATE}
 from Wine.triple_sub1
where Wine.triple_sub1.predicate=:has_color
```

설 명 : STEP 2에서 만들어진 Wine.triple_sub1 트리플 집합에서 has_color를 가지고 있는 모든 트리플들을 선택하여, 해당 트리플을 Wine.triple_sub2로 새롭게 구성한다.

결 과 :

```
ChateauChevalBlancStEmilion has_color Red, ChateauDYchemSauterne has_color Red,
ChateauLafiteRothschildPauillac has_color Red, ChateauMorgonBeaujolais has_color Red,
CorbansPrivateBinSauvignonBlanc has_color Red, ElyseZinfandel has_color Red,
FormanCabernetSauvignon has_color Red, GaryFarrellMerlot has_color Red,
StGenevieveTexasWhite has_color White, WhitehallLaneCabernetFranc has_color White,
VentanaCheninBlanc has_color White, SaucelitoCanyonZinfandel1998 has_color Rose,
SelaksSauvignonBlanc has_color White, SchlossVolradTrochenbierenausleseRiesling has_color White,
SeanThackreySiriusPetiteSyrah has_color White, SelaksIceWine has_color White,
SchlossRothermelTrochenbierenausleseRiesling has_color Rose, StonleighSavignonBlanc has_color White,
StGenevieveTexasWhite has_color White, SantaCruzMountainVineyardCabernetSauvignon has_color Rose,
.............................
```

STEP 4 : STEP 3에서 추출된 트리플 집합_sub2에서 마지막에 들어오는 arg3을 만족하는 Subject를 찾는다.

내 용 : STEP 3에서 Predicate까지 체크를 하였으며, 트리플의 마지막 구성인 Object는 트리플 집합_sub2에 대하여 질의를 수행한다.

모 듈 :

```
select {SUBJECT}
 from {TRIPLE_SET_sub2}
where {TRIPLE_SET_sub2.object=:ls_arg3}
```

설 명 : 최종적으로 트리플 집합_sub2에서 object 값과 ls_arg3를 비교하여 일치하는 트리플의 Subject가 트리플 처리 언어의 한 사이클을 통하여 얻어지는 결과값이 된다.

예 제 :

```
select {SUBJECT}
 from Wine.triple_sub2
where Wine.triple_sub2.object=:White
```

설 명 : STEP 3에서 만들어진 Wine.triple_sub2 트리플 집합에서 object 값이 White인 트리플의 Subject를 찾아서 Return하는 질의 내용이다.

결 과 :

```
CorbansPrivateBinSauvignonBlanc,CorbansSauvignonBlanc,SelaksSauvignonBlanc,StonleighSauvignonBlanc,
FormanChardonnay,MountEdenVineyardEdnaValleyChardonnay,MountadamChardonnay,PeterMccoyChardon
nay,FoxenCheninBlanc,VentanaCheninBlanc,MountadamRiesling,SelaksIceWine,CorbansDryWhiteRiesling,Sc
hlossRothermelTrochenbierenausleseRiesling,SchlossVolradTrochenbierenausleseRiesling,CongressSprings
Semillon,KalinCellarsSemillon,CortonMontrachetWhiteBurgundy,PulignyMontrachetWhiteBurgundy,StGenevi
eveTexasWhite,
```

STEP 5 : 조건 검색에 관한 정의이다. 트리플 한 사이클을 처리하기 위해서 STEP 1에서 STEP 4까지의 과정이 이루어지며, 이 과정은 조건검색 키워드 [AND, OR]에 의해서 반복적으로 이루어진다.

내 용 : 조건 검색의 수행으로 본 연구의 질의 언어를 사용하여 상세한 트리플 처리가 가능하다.

이러한 방식으로 질의 언어는 구성되며, 트리플 입력을 효율적으로 처리하기 위해 자체적인 시스템에서만 동작하도록 설계되어졌다. 그렇다면 본 시스템의 질의 처리 모듈에서 어떻게 효율적인 온톨로지 평가를 지원할 수 있는지 살펴보면 다음과 같다.

예를 들어, 사용자가 다음 질의에 대한 결과를 원한다고 가정하자.

질의 : "Find wine that hasColor red and hasBody Full and madeFromGrape Chianti"

본 모듈을 사용한 질의 인터페이스 방식에서 자연어 기반의 질의 처리 과정은 [표 17]과 같이 수행된다.

[표 17] 질의 수행 방식

①Wine(입력)[②Show Wine속성][③Show 속성의 값][④AND,OR선택] [⑤Show Wine속성][⑥Show 속성의 값][⑦AND,OR선택]...
① Wine ② hasBody ③ FULL ④ AND ⑤ hasColor ⑥ Red ...

사용자는 컨트롤된 질의 언어를 사용하여 추론모듈에서 제공된 사실 트리플을 탐색하여 쉽게 원하는 결과를 찾을 수 있다. 이러한 질의 인터페이스 방식에서는 형식화된 절차에 따라서 질의 처리가 수행되므로, 사용자의 역할은 최소화되고 대부분이 처리를 기계가 대신 수행할 수 있도록 설계하여, 일반 사용자들에 의해 용이하게 사용되어질 수 있다. 앞에서 제시한 질의를 수행하는 일련의 과정이다.

[표 18] 질의 수행 과정 상세

1. 질의 인터페이스에 사용자가 원하는 개념을 작성한다.(Wine) 2. 입력된 개념이 가지고 있는 속성들을 파악(owl:restriction)하여 객체가 가지고 있는 속성들의 리스트를 사용자에게 보여준다. 그러면 사용자는 그 중에서 사용자가 원하는 속성을 선택한다.(hasColor) 3. 선택된 속성(hasColor)이 가질수 있는 값(owl:oneOf)을 파악하여 그 값들을 사용자에게 리스트로 보여준다. 그러면 사용자는 그 중에서 원하는 속성의 값을 선택하게 된다.(red) 앞의 과정을 통하여 가장 일반적인 자연어 구문의 구조 <주어-서술어-목적어>가 만들어진다. 이렇게 만들어진 내용이 한개의 트리플이 된다. 4. 속성이 만약에 복수개의 값을 가질 수 있다면, 복수개의 값을 선택할 수 있도록 지원해 준다. 여기에서는 AND(owl:intersectionOf)와 OR(owl:unionOf)를 선택하여 사용자가 개념에 대한 좀더 상세한 정보를 입력할 수 있도록 지원해 준다. 5. 이전 과정과 동일하게 반복 진행하면서 최종 결과값을 찾게 된다.

유도된 질의 처리를 수행할 수 있도록 지원하기 위해서 시스템 내부적으로 처리해야 할 루틴이 필요하다. 클래스의 파악, 속성 파악, 속성의 값 파악, 인스턴스 파악, AND/OR 파악, Domain/Range 파악 등의 내부적인 처리모듈과 RDF(S)와 OWL의 이론에 기반한 규칙들을 적용하여 이러한 처리들을 수행하기 위한 모듈들이 필요하다. 시스템 내부적으로 처리되는 모듈들을 정리하면 다음과 같다.

1. 사용자에 의해 가장 먼저 입력되는 것은 클래스이다. 그러므로 사용자에 의해 특정의 클래스 명이 입력되면 다음과 같은 질의가 내부적으로 처리된다. 클래스의 선택은 추론된 트리플 사실에서 주어(subject)가 선택된다는 의미이다. 이에 질의 처리 모듈에서는 주어가 선택되었을 때 내부적으로 주어가 가지고 있는 관계들을 파악하고, 관계에 해당하는 추론 모듈을 통하여 주어와 연관된 모든 관련 내용들을 처리할 수 있는 준비 작업을 한다.

```
[주어가 선택되었을 때]
Find_relation(subject)
If subject has relation
  reasoning_rule2 = :reasoning_rule1(relation)
  Loop reasoning_rule2
     Is_rea_concept[ ] = result(triple.concept)
  End Loop
End If
```

2. 질의 처리 엔진에서는 내부적으로 입력된 Wine이 가지고 있는 속성들을 모두 파악하여 사용자에게 리스트로 보여준다. 이때, RDF(S)와 OWL의 일반규칙이 적용되는데, owl:Restriction으로 정의된 부분이 바로 해당 클래스의 속성을 표현하고 있다. 속성의 선택은 추론된 트리플 사실에서 서술어(relation)가 선택된다는 의미이다. 이에 질의 처리 모듈에서는 서술어가 선택되었을 때 내부적으로 서술어가 가지고 있는 관계들을 파악하고, 관계에 해당하는 추론 모듈을 통하여 서술어와 연관된 모든 관련 내용들을 처리할 수 있는 준비 작업을 한다.

```
[서술어가 선택되었을 때]
Find_relation(relation)
If relation has relation
  reasoning_rule2 = :reasoning_rule1(relation)
  Loop reasoning_rule2
         Is_rea_relation[ ] = result(triple.relation)
  End Loop
End If
Find_restriction(relation)
If relation has restriction
  Loop
    Is_rea_restriction[ ] = result(triple.restriction)
  End Loop
End If
```

3. 속성의 목록은 사용자에게 제공되고, 사용자는 목록에서 원하는 속성을 선택하게 된다. 이렇게 선택된 속성(ex, hasColor)에 대하여 내부적으로 속성의 값이 파악되고 사용자가 선택할 수 있도록 리스트를 보여주게 된다. 여기에도 RDF(S)와 OWL의 일반규칙이 적용되며, 해당 속성에 대한 값은

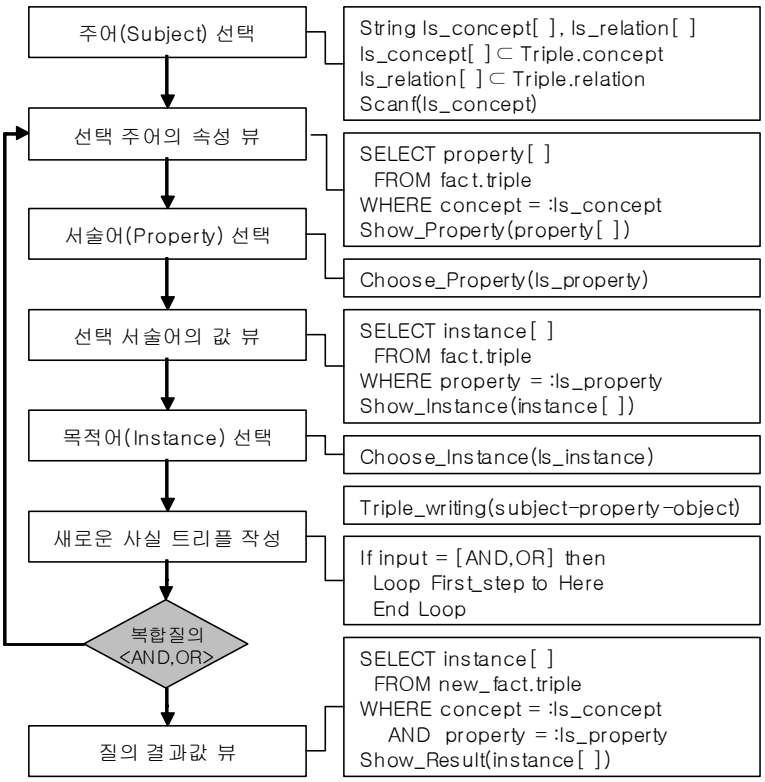
owl:oneOf로 정의하고 있다. 인스턴스의 선택은 트리플에서의 목적어(object)가 선택된 것과 같은 의미이며, 이는 앞서 설명한 해당 객체 선택 시 수행되는 추론 모듈이 다음과 같이 수행된다.

```
[목적어가 선택되었을 때]
Find_relation(object)
If object has relation
  reasoning_rule2 = :reasoning_rule1(relation)
  Loop reasoning_rule2
    Is_rea_concept1[ ] = result(triple.concept)
  End Loop
End If
```

4. 앞의 과정에서 한번의 트리플 처리를 통하여 얻은 결과는 특정의 예비 영역에 저장되어지게 되며, 트리플은 다음의 내부적 처리에 의해 저장된다.

5. 이러한 방식으로 한번의 트리플이 처리되고, 계속해서 질의를 처리할 때는 AND/OR의 처리를 통하여 사용자는 좀 더 상세하게 질의를 수행할 수 있다. 이러한 과정을 반복하여 사용자는 최종 결과를 찾는다.

이와 같은 질의 처리를 거쳐 얻어진 최종 결과는 사용자에게 제공되며, 사용자는 이들 결과 중에서 원하는 결과를 선택한다. [그림 19]는 질의 처리 모듈에서 수행되는 전체적인 처리 프로세스를 설명하고 있다. 각각의 과정에서 앞서 설명한 추론 모듈이 내부적으로 수행되어, 해당 질의에 대한 추론된 결과를 사용자에게 최종적으로 제공할 수 있다.

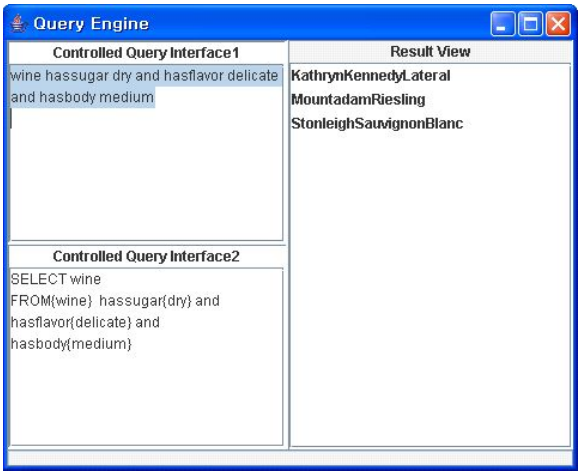


[그림 19] 질의 처리 모듈의 전체적인 수행 프로세스

[그림 19]는 질의 처리 과정의 일련의 프로세스를 보여주고 있다. 컨트롤된 질의 처리 기반 온톨로지 탐색을 하기 위해서는 사람이 하는 일련의 사고처리 프로세스를 기계가 대신 수행할 수 있도록 설계해주어야 한다. 본 연구에서는 기계의 수행범위를 넓히고 사람의 수행 범위를 최소화하여 기계가 온톨로지를 탐색하고 추론하여 온톨로지 중심의 질의 처리가 가능하도록 하였다. 또한 온톨로지에 대한 효율적 질의 처리를 위하여 추론 규칙 기반 파싱을 통한 트리플 파일을 생성하는 부분과 이러한 트리플 파일에 대하여 효율적 질의가 가능한 질의 처리 부분을 통합하여 질의 처리 모듈을 설계하였으며, 기존의 질의수행 방식의 복잡함을 해결하고자 형식화된 질의 입력 방식을 채용하여 좀 더 쉽고 정확한 질의 처리가 가능하도록 하였다. 기존 대부분의 온톨로지 질의 처리 시스템에서는 단순한 온톨로지 질의 언어에 기반하여 온톨로지내의 구조적 사실들에 대한 질의 결과를 사용자에게 제공하고 있다. 그렇지만, 온톨로지의 사용 목적인 추론에 대한 결과

를 제공하지 못하여 사용자에게 온톨로지 사용에 대해 어필하지 못하고 있다. 이에 반해, 본 연구의 질의 모듈은 추론 규칙 기반으로 얻어진 트리플 사실들을 사용하여 질의 처리가 이루어지고 있다.

본 연구의 질의 처리 방법론 평가를 위해 W3C에서 만든 와인 온톨로지(wine.rdf)를 실험 대상으로 내용 평가를 수행하였다. 먼저 와인 온톨로지는 파싱을 거쳐 총 1505개의 트리플 사실들이 생성되었으며, 1505개의 트리플들은 온톨로지 언어의 구조적 특징에 기반한 Fact 트리플 798개와 온톨로지 어휘의 규칙에 기반한 Semantic 트리플 707개로 구성되어 있다. 생성된 트리플들에 기반하여 텍스트 기반의 질의 처리 인터페이스에서 질의를 수행하는 실행화면은 [그림 20]과 같다.



[그림 20] 질의 처리 모듈 실행 화면

[그림 20]의 내용은 평가를 위해 테스트한 질의 내용에 대한 결과들을 보여주고 있다. 인터페이스 1과 같은 텍스트 형식의 자연어와 같은 질의 처리가 가능하며, 인터페이스 2에서는 기존의 질의 처리 방식의 구조적 형식을 보여주고 있다. 결과 화면에서 질의에 대한 결과들을 보여주며, 쉬운 결과 확인을 통하여 사용자들은 자신이 구축한 온톨로지에 대한 평가가 용이하게 된다. 결론적으로, 온톨로지 처리의 통합화와 쉬운 질의 처리 환경을 제공함으로써 온톨로지 사용의 범용성을 높이고, 질의 모듈의 효과로 온톨로지의 쉬운 평가를 통한 높은 활용성을 기대할 수 있다.

제 4 장 구현 및 평가

본 연구에서는 두 가지의 관점에서 도출된 결과를 가지고 연구 내용의 평가를 꾀하였다. 첫 번째 관점이 바로 온톨로지 저작 도구 측면 특히, 본 연구에서 제안한 온톨로지 파서를 통한 파싱 속도 향상의 측면이고, 두 번째 관점은 온톨로지에 대한 의미적 질의 처리에 관한 내용이다. 본 저작도구는 효율적인 온톨로지 재활용을 위한 자동 구축 방법과 구축된 온톨로지 쉬운 평가를 위해 효율적 질의 처리 방법에 중점을 두고 연구하였다. [표 19]는 기존의 온톨로지 저작 도구들과 비교한 내용이다.

[표 19] 온톨로지 저작 도구 비교

저작 도구	편집 방식	파일 형태	자동구축	추론기능	특징	파싱속도
KAON	Graph	(.kws)Project File	·	·	웹기반 멀티 구축 지원	느림
Protege 3.0	Tree & Form	(.pprj)Project File	·	plug_in 형식지원	다양한 plug_in 지원	느림
OilEd	Tree & Form	DAML+OIL,OWL	·	FaCT 추론지원	일관성 체크 지원	보통
OntoEdit	Tree & Graph	DAML+OIL	·	·	Visualization 지원	·
OWL Editor	Text	OWL	·	·	텍스트 편집창 방식	빠름
본 저작도구 (OntoMan)	Tree & Form	OWL	지원	JESS 추론지원	트리플 기반 질의처리	빠름

[표 19]에서 간단하게 파싱속도를 언급하였다. 본 연구에서 설계한 파서의 속도는 기존의 도구들의 파서 보다 빠른 처리가 이루어짐을 확인할 수 있었다. 비교를 위해 기존 도구들이 보편적으로 채택하고 있는 Jena의 RDF 파서모듈과 본 연구의 파서 모듈을 간단하게 비교한 후, 실험을 통한 각각의 도구들에서의 온톨로지 파싱 속도와 시간당 트리플 처리량을 비교하였다.

먼저, Jena의 RDF 파서 모듈과 본 연구의 파서 방법을 비교하여 살펴보면 [표 20, 21]과 같다.

[표 20] Jena의 RDF 온톨로지 파싱 모듈

```
// list the statements in the graph
StmtIterator iter = model.listStatements();

// print out the predicate, subject and object of each statement
while (iter.hasNext()) {
    Statement stmt      = iter.next();          // get next statement
    Resource  subject   = stmt.getSubject();    // get the subject
    Property  predicate = stmt.getPredicate(); // get the predicate
    RDFNode   object    = stmt.getObject();     // get the object

    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
      System.out.print(object.toString());
    } else {
       // object is a literal
       System.out.print(" ₩"" + object.toString() + "₩");
    }
    System.out.println(" .");
}
```

[표 20]의 Jena RDF 파서방법에서는 RDF 구문을 한 개씩 읽어 들여서 이를 각각의 Subject, Predicate, Object의 저장소에 순서적으로 입력하여 트리플을 만들고 있다. 전형적으로 RDF 온톨로지가 트리플 구조이기 때문에 바로 한 개의 구문이 트리플로 구성되어져 있으므로 가장 기본적인 파싱 방법이다. 그리하여 많은 저작도구들에서는 Jena에서 제공하는 RDF 파서 방법을 채택하여 온톨로지를 파싱하고 있다. 이에 반해, 본 연구의 파싱 방법에서는 이렇게 읽어 들여온 한 개의 구문에 대해서 사전에 Token 추출의 과정을 거쳐 구문을 전체적으로 Token 단위로 분석한 후, 이를 트리플로 작성하는 방법을 채택하여 파서를 설계하였다. 자세한 파싱 처리 방법은 4장의 [그림 12]에 소개되어 있다. [표 21]은 본 연구에서

제안한 온톨로지 RDF, OWL 파싱 모듈이다.

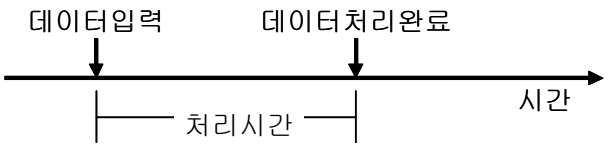
[표 21] 본 연구에서 제안한 온톨로지 파싱 모듈

```
// list the statements in the graph
StmtIterator iter = model.listStatements();

// print out the predicate, subject and object of each statement
while (iter.hasNext()) {
    Statement stmt              = iter.next();        // get next statement
    Pre_Statement pstmt         = token_extraction(); // preprocessing
    Resource  subject           = pstmt.getSubject();   // get the subject
    Property  predicate         = pstmt.getPredicate(); // get the predicate
    RDFNode   object            = pstmt.getObject();    // get the object

    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
      System.out.print(object.toString());
    } else {
       // object is a literal
       System.out.print(" ₩"" + object.toString() + "₩");
    }
    System.out.println(" .");
}
```

앞에서 비교한 각각의 파싱 모듈의 비교를 위해, 온톨로지에 대한 파싱 속도를 다음과 같은 방법으로 측정하였다. 파서의 처리 시간 측정을 위하여 기준을 데이터의 입력 시간에서 데이터 처리가 완료된 시점까지의 시간을 파싱 처리 시간으로 간주하였다.

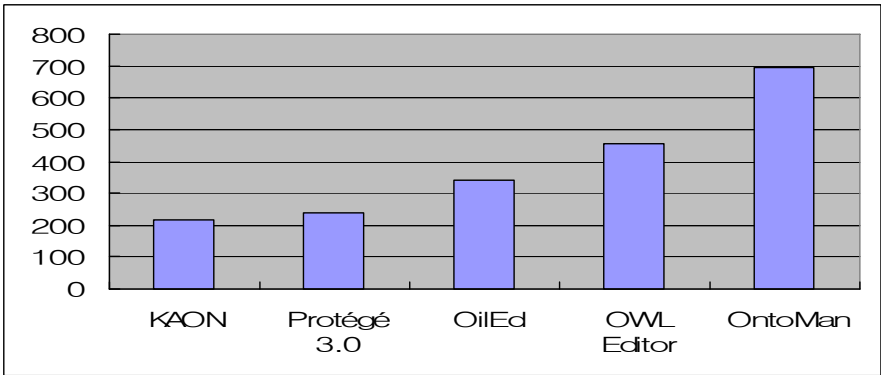


위의 파싱 처리 시간을 기준으로 각각의 저작 도구들에서 세 가지의 대표 온톨로지를 파싱하여 측정한 시간은 [표 22]와 같다.

[표 22] 온톨로지 저작 도구의 파싱 속도 (단위: 초)

저작 도구	KAON	Protege 3.0	OilEd	OWL Editor	OntoMan
Wine	4.213	3.827	2.689	1.997	1.315(0.002)
Food	2.933	2.664	1.872	1.390	0.915(0.001)
Beer	2.125	1.930	1.356	1.007	0.663(0.001)

[표 22]의 파서 처리 시간에 근거하여 각 저작도구 파서의 단위 시간당 트리플 처리량(throughput)을 측정하였으며, 그 기준은 단위시간(1초)동안 트리플 생성량을 비교하였다.



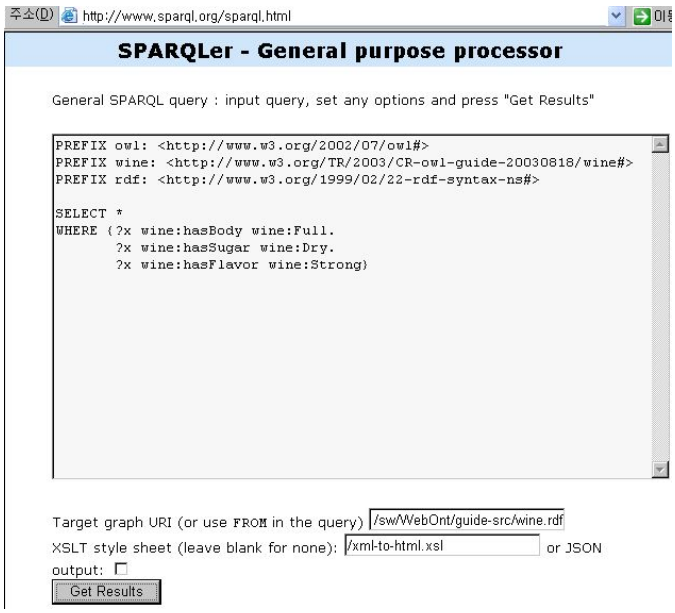
[그림 21] 온톨로지 저작 도구의 단위 시간당 트리플 처리량

본 연구의 저작 도구 설계 부분에서 온톨로지 파싱 속도를 고려하여 설계하였으며, 또한 RDF와 OWL로 작성된 파일은 모두 파싱 할 수 있도록 유연하게 파서를 설계하여, [표 22]에서 확인 할 수 있듯이, 기존의 저작도구들보다 향상된 파싱 속도를 측정할 수 있었다. 또한 모든 시스템에서 수행하는 온톨로지 구조에 대한 1차 파싱의 결과에서 뿐만 아니라, 추론을 통한 의미적 2차 파싱을 하는데 소요되는 시간은 괄호안에 작성된 내용처럼 전체 파싱의 속도에 영향을 미치지 않을 만

큼 빠르게 파싱한다.

두 번째 본 연구의 결과로 추론 규칙을 적용한 두 단계 온톨로지 파싱을 통하여 생성된 트리플 기반의 질의 처리 모듈을 통한 사용자 중심의 쉬운 질의 처리와 의미적 질의 처리 결과에 관한 내용이다. 트리플 기반의 질의 모듈을 통한 질의 수행 단계에서 처리 과정이 끝나는 시점의 결과 값을 실험을 통하여 측정하였다.

두 번째 평가의 실험에서는 RDF, OWL 온톨로지에 대한 대표적 질의 언어인 SPARQL의 질의 처리와 본 연구의 트리플 기반의 질의 처리의 내용을 비교하였다. 먼저 SPARQL과 본 연구의 질의 처리의 사용자 인터페이스를 비교하여 설명하면 다음과 같다. [그림 20]은 SPARQL을 이용하여 온톨로지 질의를 수행할 수 있도록 웹 기반 온톨로지 질의 처리 인터페이스인 SPARQLer의 질의 수행 화면이다. [그림 22]에서 확인할 수 있듯이, 질의를 위해 기본적은 SPARQL의 구조적 형식을 알고 있어야 사용자는 해당 온톨로지에 대한 질의 처리를 수행할 수 있다. 또한 질의 수행과정에서 온톨로지내의 해당 정보들에 대해서도 알고 있어야 질의 처리를 수행할 수 있다.



[그림 22] SPARQLer의 질의 수행 인터페이스 화면

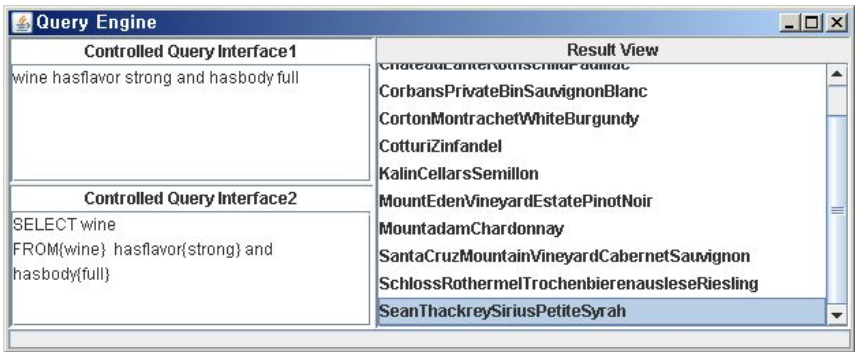
[그림 22]의 질의 수행 후, 해당 결과 값을 [그림 23]에서 보여주고 있다.

SPARQLer Query Results

x
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#CotturiZinfandel>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#CortonMontrachetWhiteBurgundy>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#CorbansPrivateBinSauvignonBlanc>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#SantaCruzMountainVineyardCabernetSauvignon>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#KalinCellarsSemillon>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#MountEdenVineyardEstatePinotNoir>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#SeanThackreySiriusPetiteSyrah>
<http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#MountadamChardonnay>

[그림 23] SPARQLer의 질의 수행 결과 값

이러한 복잡한 질의 과정을 최소화 과정으로 처리하고자 본 연구에서는 질의 인터페이스를 유도된 텍스트 기반의 질의 처리 모듈을 적용하여 구현하였다. 그리하여 사용자는 온톨로지 질의 언어 구조뿐만 아니라, 온톨로지의 내용을 이해하고 있지 않더라도 유도되는 텍스트를 따라가면서 쉽게 질의를 수행할 수 있으며, 질의 수행 결과도 쉽게 확인할 수 있도록 사용자 질의 인터페이스를 설계하였다. [그림 24]는 본 연구에서 구현한 온톨로지 질의 인터페이스 화면이다.



[그림 24] 본 연구의 질의 인터페이스 화면

앞에서 간단하게 본 연구의 질의 인터페이스를 사용하여 얻을 수 있는 온톨로지 질의 수행 시 이점에 대하여 설명하였다. 이러한 이점과 더불어, 본 연구의 질의 처리 모듈의 성과중의 하나가 바로 효율적 온톨로지내 정보 검색이다. 앞에서

설명했던 온톨로지 파싱 모듈에서 본 연구의 온톨로지 파서는 기존의 온톨로지를 빠르게 파싱하는 방법론과 더불어, 파싱된 온톨로지 트리플을 가지고 추론규칙을 적용한 재 파싱을 수행하는 방법론을 채택한 2단계 파싱을 수행하고 있다. 이렇게 2단계의 파싱을 수행하는 목적중의 하나가 바로 기존의 질의 언어들이 채택하고 있는 단순한 구조 기반의 온톨로지 질의 처리에서 의미기반의 온톨로지 질의 처리를 수행하고자 하였다.

본 연구에서는 이렇게 2단계의 파싱과정을 통하여 만들어진 Fact 트리플 집합과 Semantic 트리플 집합에 기반하여 질의 처리를 수행하고 있다. 이러한 과정에 의거하여 본 연구의 질의 처리의 수행 결과값은 기존의 질의 처리 결과값보다 향상된 온톨로지내 정보 추출이 가능함을 확인할 수 있었다. [표 23]은 대표적인 온톨로지 질의 언어인 SPARQL을 채택하여 온톨로지에 대한 질의 처리를 수행하는 SPARQLer와 본 연구의 질의 처리 방법을 통하여 Wine.rdf 온톨로지에 대한 다양한 질의에 대한 결과를 보여주고 있다.

[표 23] 질의 처리 결과 비교

질 의 내 용	Wine.rdf	SPARQLer	본 연구 질의 처리 모듈
	기대값	결과값	결과값
SELECT * WHERE {?x wine:hasBody wine:Medium. ?x wine:hasColor wine:White. ?x wine:hasFlavor wine:Strong}	2	0	1
SELECT * WHERE {?x wine:hasBody wine:Full}	16	13	16
SELECT * WHERE {?x wine:madeFromGrape wine:SemillonGrape}	3	1	3
SELECT * WHERE {?x wine:hasBody wine:Full. ?x wine:hasColor wine:White. ?x wine:hasFlavor wine:Strong. ?x wine:madeFromGrape wine:SemillonGrape}	1	0	1
SELECT * WHERE {?x wine:hasFlavor wine:Strong. }	15	12	15
SELECT * WHERE {?x wine:hasFlavor wine:Strong. ?x wine:hasBody wine:Full}	11	9	11
SELECT * WHERE {?x wine:hasFlavor wine:Strong. ?x wine:hasBody wine:Full. ?x wine:hasSugar wine:Dry}	8	8	8
SELECT * WHERE {?x wine:hasFlavor wine:Strong. ?x wine:hasBody wine:Full. ?x wine:hasSugar wine:Dry. ?x wine:madeFromGrape wine:SemillonGrape}	1	0	0

실험 결과 내용에서 확인할 수 있듯이, 본 연구에서 온톨로지내의 정보는 텍스트 기반의 사용자 중심 질의 인터페이스를 통하여 쉽고 빠르게 질의 처리가 가능하며, SPARQLer의 인터페이스 화면과 비교해서 확인 할 수 있다. 그리고 추론 규칙을 적용한 트리플 파서에서 추출된 트리플을 기반으로 좀 더 정확하고 상세한 온톨로지 탐색이 가능함을 확인할 수 있었다. [표 23]의 내용에서 검색율은 와인 온톨로지 내의 총 해당 결과에서 각각의 질의 방법을 통하여 구해진 결과값을 비교하였다. 그 결과 본 연구의 질의 모듈에서의 검색율은 약 0.91이고, SPARQL

는 약 0.76의 검색율을 측정할 수 있었다. 그리하여 본 연구의 질의 모듈에서 기존 SPARQL 기반의 질의 처리 방법에서의 검색율보다 약 0.15정도의 향상된 확인 할 수 있었다. 그리고 [표 23]의 내용에서도 확인할 수 있듯이, 검색된 결과는 100% 원하는 결과내의 값이었음이 확인되었다. 또한 평가에 포함된 질의 시간은 보통 본 연구의 질의 인터페이스에서는 한 개의 트리플 질의를 생성하는데 약 5초 정도가 소요되었고, 기존의 SPARQL에서의 형식화된 구조를 모르더라도 유도된 텍스트 질의 인터페이스 방식으로 질의 수행의 오차율을 최소화 하였다. 또한 모든 질의에 대하여 질의의 길이에 상관없이 해당 질의의 결과값을 확인하는데 1초 미만의 시간이 소요되었다. 결론적으로 본 연구의 온톨로지 처리 기술은 향후 시맨틱 웹의 실현을 위해 반드시 요구되는 기술이라 할 수 있다.

제 5 장 결론 및 제언

본 연구는 시맨틱 웹 환경에서의 정보검색을 위한 효율적 온톨로지 처리 기술에 관한 연구로써, 효율적인 온톨로지 처리를 위해 기존 온톨로지에 대한 파싱의 방법을 개선하여 파싱의 속도를 향상시켰으며, 또한 기존의 질의 언어 방법을 트리플 기반의 사용자 중심의 질의 처리가 가능하도록 설계 구현함으로써 온톨로지 질의 언어나 온톨로지의 내용을 모르는 일반 사용자들도 쉽게 온톨로지에 대한 질의를 수행할 수 있도록 하였다. 또한 질의 처리의 내용에서 기존의 온톨로지 구조에 기반한 질의 처리에서 구조뿐만 아니라 온톨로지 어휘들이 포함하고 있는 추론의 내용들을 반영하여 질의할 수 있도록 온톨로지에 대한 구조적 파싱과 파싱된 내용을 추론 규칙을 적용하여 재 파싱을 통한 의미적 트리플 생성 작업을 거쳐 온톨로지에 대한 좀 더 상세하고 정확한 질의 수행이 가능하도록 기능을 향상 시켰다. 그리고 온톨로지의 재사용을 위해 기 구축된 지식 베이스인 워드넷을 이용하여 사용자는 쉽게 원하는 주제에 대한 프레임 온톨로지 구축할 수 있었으며, 이 과정에서 사용자는 개념의 수집과 수집된 개념간의 관계의 과정에 대해서 자동으로 구축이 가능하게 되었다. 그 결과, 사용자는 온톨로지 구축의 시간을 단축할 수 있으며, 지금까지 표준적인 모형없이 구축되어져 온 온톨로지의 표준화를 기대할 수 있다.

본 연구에서 제안하는 의미적 정보검색을 위한 효율적 온톨로지 처리 기술은 온톨로지 자동 구축 및 효율적 질의 처리 방법이다. 먼저 온톨로지 자동 구축을 위해 기존의 지식을 효율적으로 재사용할 수 있는 방법을 설계하여, 쉽게 자신이 원하는 도메인에 대한 프레임 온톨로지를 구축할 수 있다. 하지만 자동 구축의 방법에서 기존의 지식에 기반하여 자동 구축을 수행함으로써, 기존 지식에서 포함하고 있는 내용의 수준에 의해 자동 구축 결과가 좌우된다. 또한 속성 및 인스턴스의 자동 생성의 부분은 처리할 수 없다. 이러한 부분들은 향후 다양한 지식을 연계하여 자동 구축을 위한 지식 베이스를 마련하거나 개념에 대한 내용을 자동으로 웹에서 수집하여 관련된 개념들과 인스턴스들을 가공하는 과정을 추가하여

좀더 향상된 자동 온톨로지 구축 방법을 모색하여야 하겠다. 또한 효율적 온톨로지 질의 처리 방법에서는 기존의 질의 처리 방법에서 수행하는 온톨로지의 단순한 구조에 기반한 질의 처리에서 온톨로지의 내용에 기반한 의미적 질의 처리를 수행하였다는 의의가 있다. 또한 사용자 중심의 질의 처리 인터페이스를 설계하여 사용자는 쉽게 구축한 또는 구축된 온톨로지를 평가하여 자신이 원하는 온톨로지로 재가공할 수 있다. 하지만 질의 처리의 수행 과정에서 기존 질의 언어에서 지원하는 다양한 기능들을 모두 지원하게 되면 사용자 중심의 질의 인터페이스 설계가 더욱 복잡해지는 결과를 초래하여, 본 연구의 온톨로지 질의 처리는 추론된 의미적 트리플을 사용하여 의미적 온톨로지 질의 처리를 수행하는 반면 복잡하고 상세한 질의 처리 수행은 불가능하다. 이러한 부분 역시 향후 연구되어야 할 연구 과제이다.

온톨로지 연구 분야에서 효율적 온톨로지 처리 기술 연구가 요구되며, 온톨로지 구축을 위해 요구되는 기술, 온톨로지 공유를 위한 통합(Integration) 표준 방식에 대한 연구 그리고 온톨로지 효율적 처리를 위한 추론 기술 및 질의 처리 기술들을 개발함으로써, 온톨로지 연구에 대한 기술적 향상에 도움이 된다. 끝으로, 온톨로지를 웹 상의 정보검색에 응용한 온톨로지 기반 시맨틱 정보검색은 새로운 차원의 선택도를 갖는 의미적 정보검색을 가능하게 되었다.

참고문헌

[1] 이재호, “시맨틱 웹의 온톨로지 언어”, 정보과학회지, 제21권, 제3호, pp. 18-27, (2003).

[2] 양정진, “시맨틱 웹에서의 온톨로지 공학”, 정보과학회지, 제21권, 제3호, pp.28-35, (2003).

[3] Broekstra, J., Kampman, A., and van Harmelen, F. : Sesame:A Generic Architecture for Storing and Querying RDF and RDF Schema. ISWC2002, page 54-68, (2002)

[4] Seaborne, A. : RDQL A Query Language for RDF. W3C member submission, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109, 2004.

[5] Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, (2001).

[6] Maedche A., Ontology Learning for the Semantic Web, Academic Publishers, (2002).

[7] Davies, J., Fensel, D., Harmelen, F.V., Towards The Semantic Web, JOHN WILEY & SON Ltd, (2003).

[8] Fensel, D., Hendler, J., Lieberman, H., Wahlster, W., Spinning the Semantic Web, The MIT Press, (2003).

[9] Michael Denny, "Ontology editor survey results", http://xml.com/2002/11/06/Ontology_Editor_Survey.html/, (2002).

[10] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, "Jena : Implementing the Semantic Web Recommendations", HP Laboratories Technical Report HPL-2003-146, (2003).

[11] "Jena 2 - A Semantic Web Framework",http://www.hpl.hp.com/semweb/jena.htm

[12] "F-OWL: An OWL Inference Engine in Flora-2", http://fowl.sourceforge.net/about.html/.

[13] "OWL: Hoolet", http://owl.man.ac.uk/hoolet/.

[14] Fabien L. Gandon, Norman M. Sadeh, "OWL Inference Engine in Jess", http://www.cs.cmu.edu/~sadeh/MyCampusMirror/OWLEngine.html.

[15] “Jess, the Rule Engine for the Java Platform", http://herzberg.ca.sandia.gov/jess/References.

[16] Jose Saias, Paulo Quaresma, "A methodology to create ontology-based information retrieval systems", EPIA2003, pp. 424-434, (2003).

[17] By Jian Wen, Zhoujun Li, Xiaohua Hu, "Ontology Based Clustering for Improving Genomic IR", CBMS2007, pp. 225-230, (2007).
[18] Fellbaum, C.: WordNet: An Electronic Lexical Database. Cambridge: MIT Press, (1999).
[19] Sudip Kumar Naskar, Sivaji Bandyopadhyay, "Word Sense Disambiguation Using Extended WordNet", ICCTA2007, pp. 446-450, (2007).
[20] Ziv Baida, Jaap Gordijn, Hanne Saele, Hans Akkermans, Andrei Z. Morch, "An Ontological Approach for Eliciting and Understanding Needs in e-Services", CAiSE2005, pp. 400-414, (2005).
[21] Maedche A, Zacharias V. Clustering ontology-based metadata in the Semantic Web. In Proceedings of the Joint Conferences 13th European Conference on Machine Learning (ECML'02) and 6th European Conference on Principles and Practive of Knowledge Discovery in Databased (PKDD'02). Springer, LNAI: Helsinki, Finland, (2002).
[22] Shuang Liu, Fang Liu, Clement Yu, Weiyi Meng, "An effective approach to document retrieval via utilizing WordNet and recognizing phrases", SIGIR2004, pp. 68-77, (2004).
[23] De Luca, E. W., Nurnberger, A., Ontology-based Semantic Online Classification of Documents: Supproting Users in Searching the Web, In: Proc. of the European Symposium on Intelligent Technologies (EUNITE 2004), Aachen (2004).
[24] Yohan Jin, Latifur Khan, Lei Wang, Mamoun Awad, "Image annotations by combining multiple evidence & wordNet", MULTIMEDIA 2005, pp. 446-450, (2005).
[25] Latifur Kahn, Feng Luo, "Ontology Construction for Information Selection", Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence, pp. 122-127, (2002).
[26] Gloria L. Zuniga, "Ontology: its transformation from philosophy to information systems", Proceedings of the international conference on Formal Ontology in Information System, pp. 187-197, (2001).
[27] Marie-Helene Abel, Dominique Lenne, Claude Moulin, Ahcene Benayache, "Using Two Ontologies to Index e-Learning Resources", WI'04, pp. 549-552, (2004).

[28] Natalya F. Noy and Deborah L. McGuinness, A Guide to Creating Your First Ontology, Stanford University.
[29] http://www.ontoknowledge.org/tools/ontoedit.shtml.
[30] http://protege.stanford.edu/.

부록 1. 추론 규칙 및 규칙에 대한 파싱 모듈 설계도

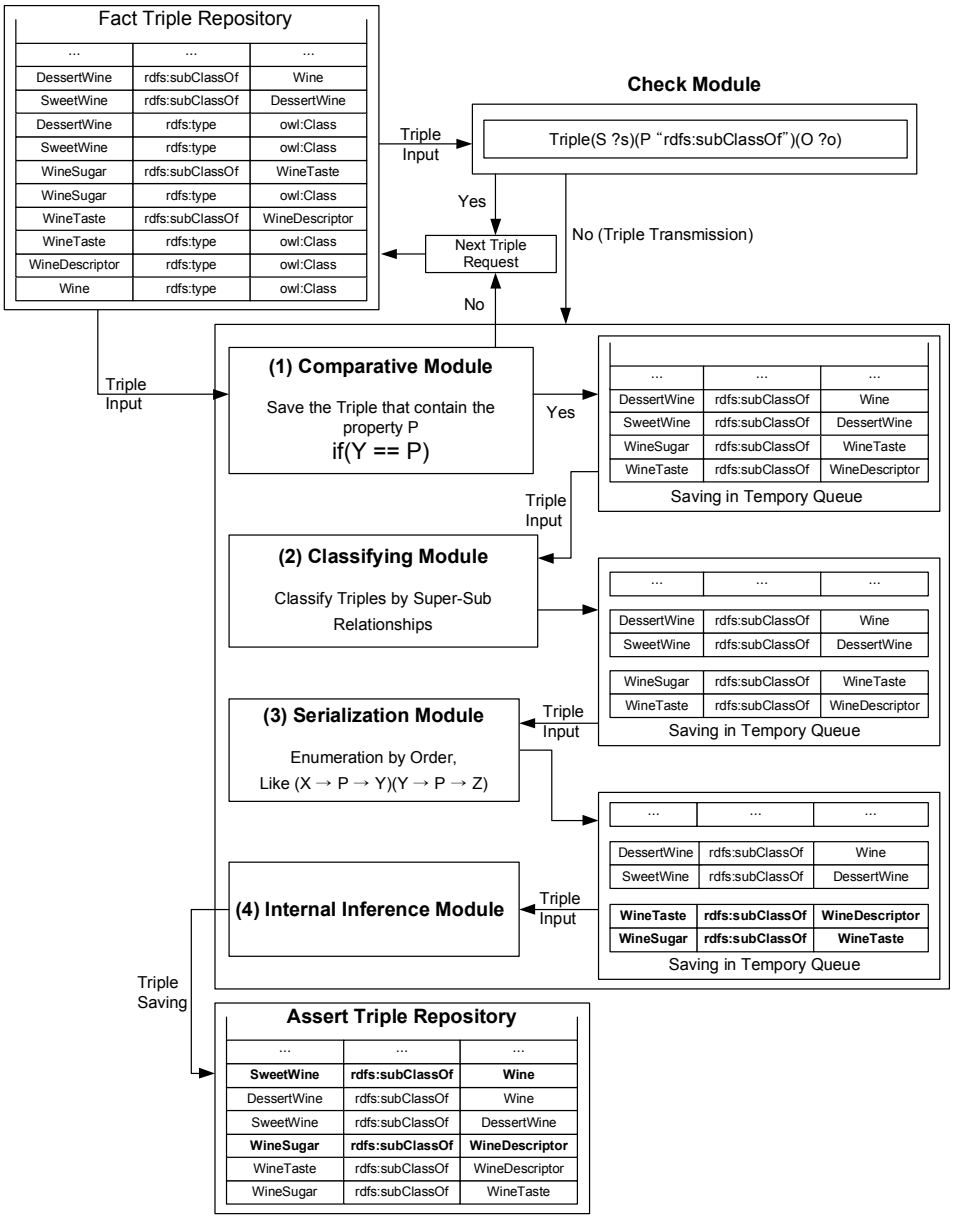
1. 추론 규칙

규칙이름		추론 방식
1	Propagate Transitive Properties	Triple(S ?p)(P rdf:type)(O owl:TransitiveProperty) Triple(S ?x)(P ?p)(O ?y) Triple(S ?y)(P ?p)(O ?z) => Assert Triple(S ?x)(P ?p)(O ?z)
2	Complete Typing of instances of objects with inherited types	Triple(S ?y)(P rdfs:subClassOf)(O ?z) Triple(S ?x)(P rdf:type)(O ?y) => Assert Triple(S ?x)(P rdf:type)(O ?z)
3	Complete typing of instances of properties with inherited types	Triple(S ?y)(P rdfs:subPropertyOf)(O ?z) Triple(S ?a)(P ?y)(O ?b) => Assert Triple(S ?a)(P ?z)(O ?b)
4	Default behavior for signature propagation	Triple(S ?x)(P rdfs:subPropertyOf)(O ?y) Triple(S ?y)(P rdfs:domain)(O ?z) => Assert Triple(S ?x)(P rdfs:domain)(O ?z)
		Triple(S ?x)(P rdfs:subPropertyOf)(O ?y) Triple(S ?y)(P rdfs:range)(O ?z) => Assert Triple(S ?x)(P rdfs:range)(O ?z)
5	Complement with instances of symmetric properties	Triple(S ?p)(P rdf:type)(O owl:SymmetricProperty) Triple(S ?x)(P ?p)(O ?y) => Assert Triple(S ?y)(P ?p)(O ?x)
6	Complete with instances of inverse properties	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?x)(P ?p1)(O ?y) => Assert Triple(S ?y)(P ?p2)(O ?x)
7	The inverse property of a functional property is an inverse functional property	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?p1)(P rdf:type)(O owl:FunctionalProperty) => Assert Triple(S ?p2)(P rdf:type)(O owl:inverseFunctionalProperty)
8	The inverse property of an inverse functional property is a functional property	Triple(S ?p1)(P owl:inverseOf)(O ?p2) Triple(S ?p1)(P rdf:type)(O owl:inverseFunctionalProperty) => Assert Triple(S ?p2)(P rdf:type)(O owl:FunctionalProperty)
9	Complete with instances of equivalent Class	Triple(S ?t1)(P owl:equivalentClass)(O ?t2) Triple(S ?x)(P rdf:type)(O ?t1) => Assert Triple(S ?x)(P rdf:type)(O ?t2)

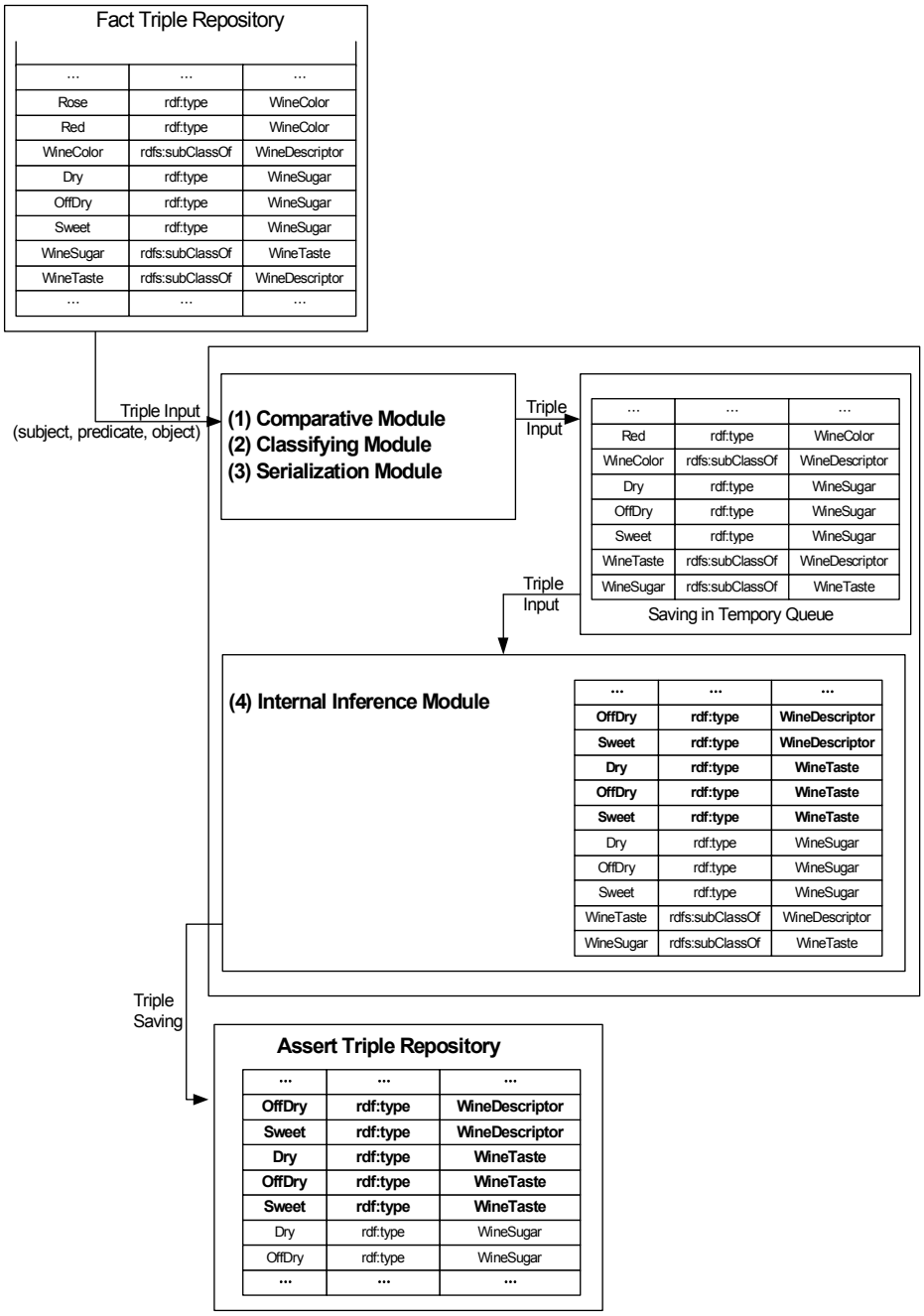
10	Complete hierarchy with equivalent classes: if t1<=>t2 then t1<t2, and by symmetry we also get t2<t1	Triple(S ?t1)(P owl:equivalentClass)(O ?t2) => Assert Triple(S ?t1)(P rdfs:subClassOf)(O ?t2) => Assert Triple(S ?t2)(P rdfs:subClassOf)(O ?t1)
11	Members of disjoint classes are different instances	Triple(S ?c1)(P owl:disjointWith)(O ?c2) Triple(S ?o1)(P rdf:type)(O ?c1) Triple(S ?o2)(P rdf:type)(O ?c2) => Assert Triple(S ?o1)(P owl:differentFrom)(O ?o2)
12	Two classes subsuming each other are equivalent	Triple(S ?c1)(P rdfs:subClassOf)(O ?c2) Triple(S ?c2)(P rdfs:subClassOf)(O ?c1) Test(nnq ?c1 ?c2) => Assert Triple(S ?c1)(P owl:equivalentClass)(O ?c2)
13	Complete instances with properties of equivalent instances	Triple(S ?s1)(P owl:sameIndividualAs)(O ?s2) Triple(S ?s1)(P ?p)(O ?o) => Assert Triple(S ?s2)(P ?p)(O ?o)
14	Complete instances of properties with instances of equivalent properties	Triple(S ?p1)(P owl:equivalentProperty)(O ?p2) Triple(S ?s)(P ?p1)(O ?o) => Assert Triple(S ?s)(P ?p2)(O ?o)
15	Complete hierarchy with equivalent properties: if p1<=>p2 then p1<p2, and by symmetry we also get p2<p1	Triple(S ?p1)(O owl:equivalentProperty)(O ?p2) => Assert Triple(S ?p1)(P rdfs:subPropertyOf)(O ?p2) => Assert Triple(S ?p2)(P rdfs:subPropertyOf)(O ?p1)
16	Two properties subsuming each other are equivalent	Triple(S ?p1)(P rdfs:subPropertyOf)(O ?p2) Triple(S ?p2)(P rdfs:subPropertyOf)(O ?p1) Test(neq ?p1 ?p2) => Assert Triple(S ?p1)(P owl:equivalentProperty)(O ?p2)
17	Derive equivalence form multiple instantiations of a functional property with the same subject	Triple(S ?p)(P rdf:type)(O owl:FunctionalProperty) Triple(S ?s)(P ?p)(O ?o1) Triple(S ?s)(P ?p)(O ?o2) Test(neq ?o1 ?o2) => Assert Triple(S ?o1)(P owl:sameIndividualAs)(O ?o2)
18	Derive equivalence from multiple instantiations of inverse functional property with the same object	Triple(S ?p)(P rdf:type)(O owl:InverseFunctionalProperty) Triple(S ?s1)(P ?p)(O ?o) Triple(S ?s2)(P ?p)(O ?o) Test(neq ?s1 ?s2) => Assert Triple(S ?s1)(P owl:sameIndividualAs)(O ?s2)
19	Complete the extension of the Thing class with all other classes	Triple(S ?s)(P rdf:type)(O ?o) => Assert Triple(S ?s)(P rdf:type)(O owl:Thing)
20	A property inverse of it-self is symmetric	Triple(S ?p)(P owl:inverseOf)(O ?p) Assert Triple(S ?p)(P rdf:type)(O owl:SymmetricProperty)

2. 추론 규칙에 대한 파싱 모듈 설계도

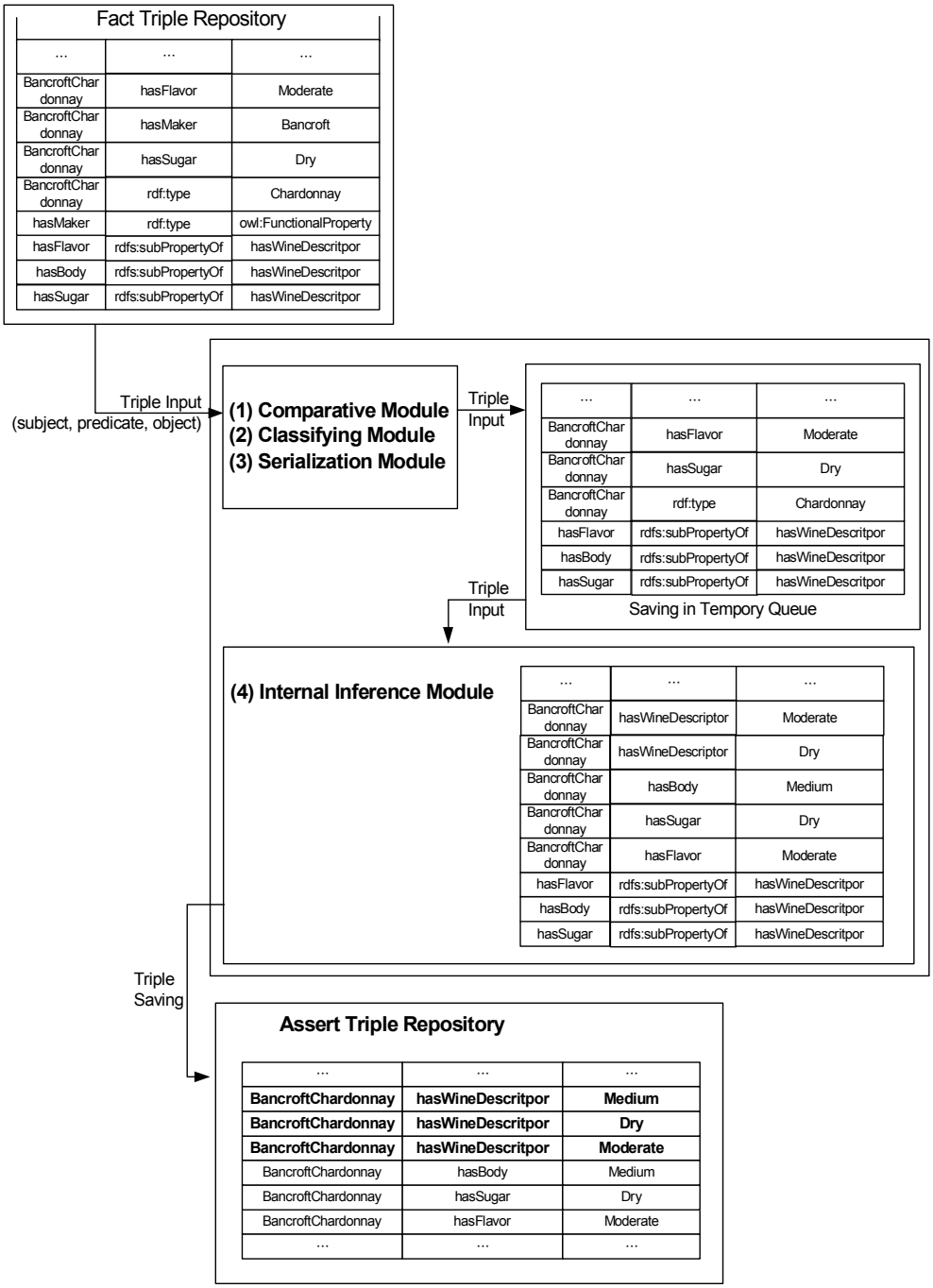
2.1 규칙 1



2.2 규칙 2


Fact Triple Repository
Triple Input
(subject, predicate, object)
(1) Comparative Module
(2) Classifying Module
(3) Serialization Module
Triple Input
Saving in Tempory Queue
Triple Input
(4) Internal Inference Module
Triple Saving
Assert Triple Repository

2.3 규칙 3



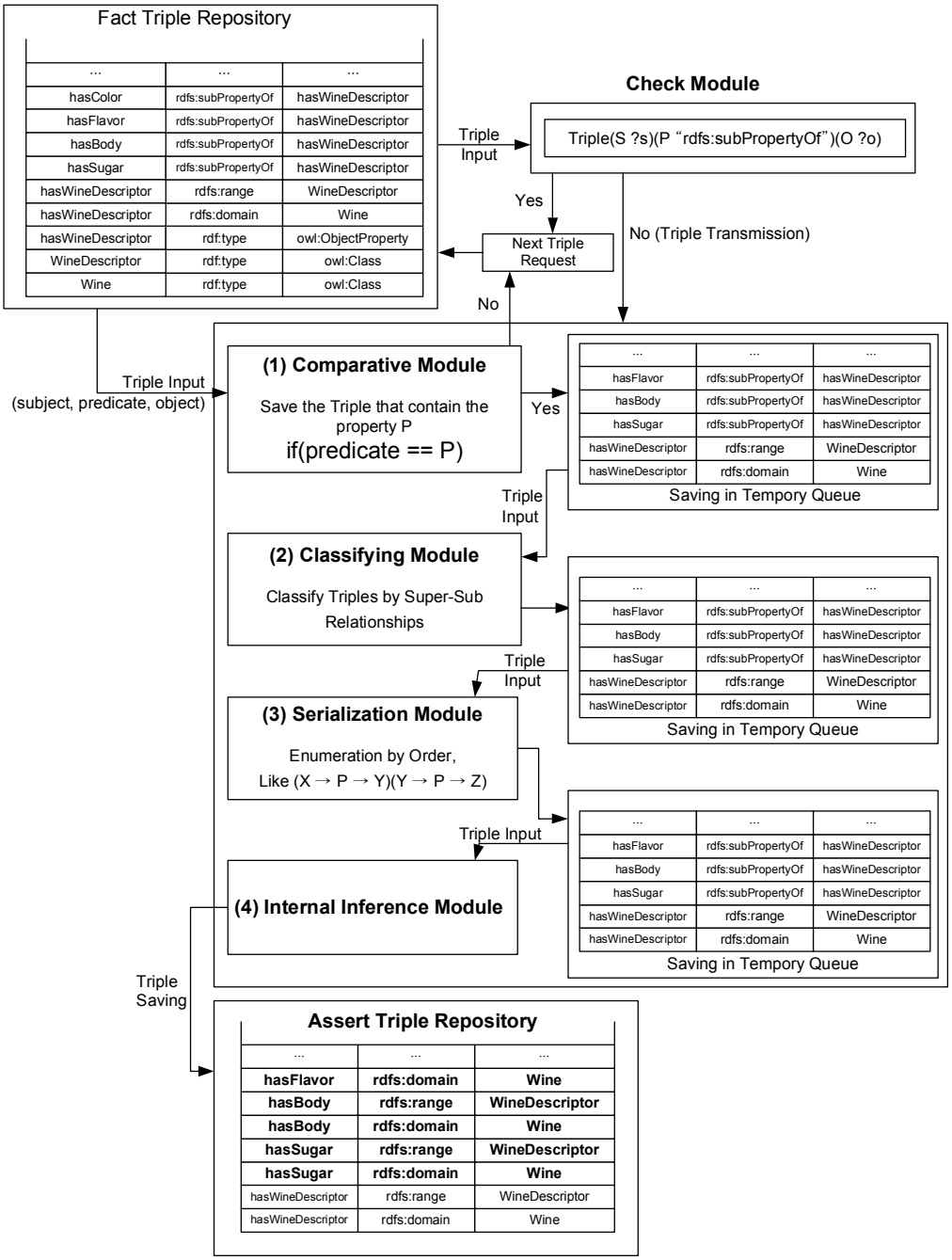
...	...	...
BancroftChardonnay	hasFlavor	Moderate
BancroftChardonnay	hasMaker	Bancroft
BancroftChardonnay	hasSugar	Dry
BancroftChardonnay	rdf:type	Chardonnay
hasMaker	rdf:type	owl:FunctionalProperty
hasFlavor	rdfs:subPropertyOf	hasWineDescritpor
hasBody	rdfs:subPropertyOf	hasWineDescritpor
hasSugar	rdfs:subPropertyOf	hasWineDescritpor

...	...	...
BancroftChardonnay	hasFlavor	Moderate
BancroftChardonnay	hasSugar	Dry
BancroftChardonnay	rdf:type	Chardonnay
hasFlavor	rdfs:subPropertyOf	hasWineDescritpor
hasBody	rdfs:subPropertyOf	hasWineDescritpor
hasSugar	rdfs:subPropertyOf	hasWineDescritpor

...	...	...
BancroftChardonnay	hasWineDescriptor	Moderate
BancroftChardonnay	hasWineDescriptor	Dry
BancroftChardonnay	hasBody	Medium
BancroftChardonnay	hasSugar	Dry
BancroftChardonnay	hasFlavor	Moderate
hasFlavor	rdfs:subPropertyOf	hasWineDescritpor
hasBody	rdfs:subPropertyOf	hasWineDescritpor
hasSugar	rdfs:subPropertyOf	hasWineDescritpor

...	...	...
BancroftChardonnay	**hasWineDescritpor**	**Medium**
BancroftChardonnay	**hasWineDescritpor**	**Dry**
BancroftChardonnay	**hasWineDescritpor**	**Moderate**
BancroftChardonnay	hasBody	Medium
BancroftChardonnay	hasSugar	Dry
BancroftChardonnay	hasFlavor	Moderate
...	...	...

2.4 규칙 4



...	...	...
hasColor	rdfs:subPropertyOf	hasWineDescriptor
hasFlavor	rdfs:subPropertyOf	hasWineDescriptor
hasBody	rdfs:subPropertyOf	hasWineDescriptor
hasSugar	rdfs:subPropertyOf	hasWineDescriptor
hasWineDescriptor	rdfs:range	WineDescriptor
hasWineDescriptor	rdfs:domain	Wine
hasWineDescriptor	rdf:type	owl:ObjectProperty
WineDescriptor	rdf:type	owl:Class
Wine	rdf:type	owl:Class

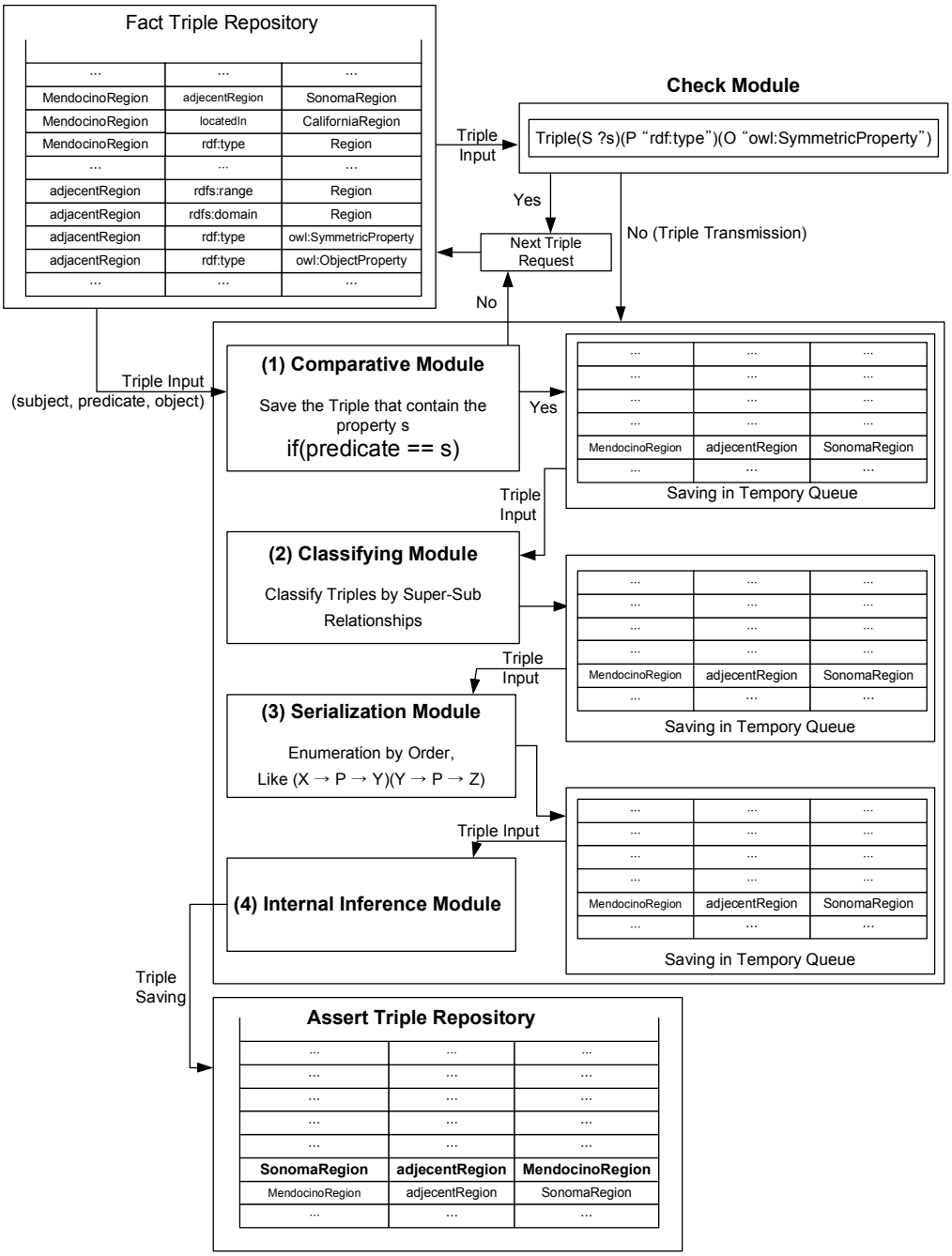
...	...	...
hasFlavor	rdfs:subPropertyOf	hasWineDescriptor
hasBody	rdfs:subPropertyOf	hasWineDescriptor
hasSugar	rdfs:subPropertyOf	hasWineDescriptor
hasWineDescriptor	rdfs:range	WineDescriptor
hasWineDescriptor	rdfs:domain	Wine

...	...	...
hasFlavor	rdfs:subPropertyOf	hasWineDescriptor
hasBody	rdfs:subPropertyOf	hasWineDescriptor
hasSugar	rdfs:subPropertyOf	hasWineDescriptor
hasWineDescriptor	rdfs:range	WineDescriptor
hasWineDescriptor	rdfs:domain	Wine

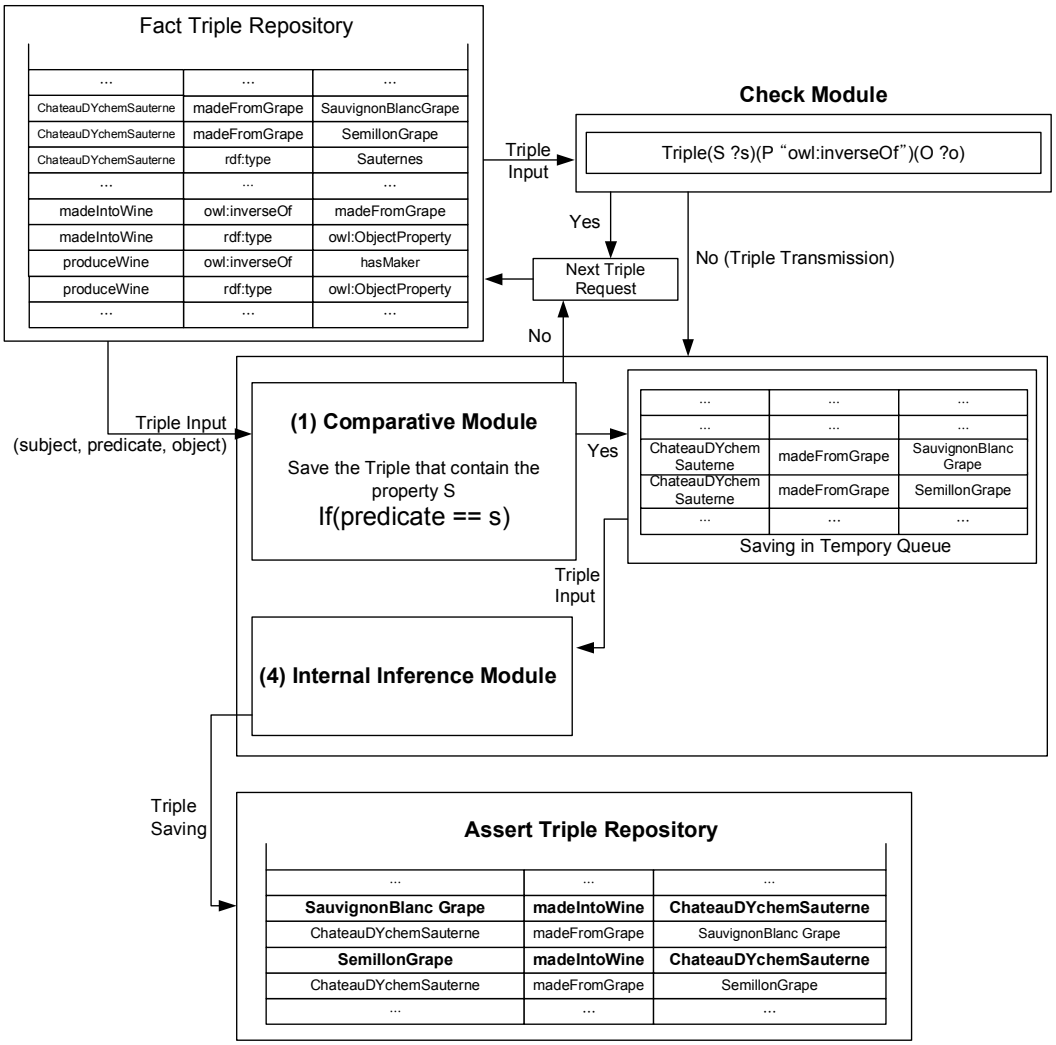
...	...	...
hasFlavor	rdfs:subPropertyOf	hasWineDescriptor
hasBody	rdfs:subPropertyOf	hasWineDescriptor
hasSugar	rdfs:subPropertyOf	hasWineDescriptor
hasWineDescriptor	rdfs:range	WineDescriptor
hasWineDescriptor	rdfs:domain	Wine

...	...	...
hasFlavor	**rdfs:domain**	**Wine**
hasBody	**rdfs:range**	**WineDescriptor**
hasBody	**rdfs:domain**	**Wine**
hasSugar	**rdfs:range**	**WineDescriptor**
hasSugar	**rdfs:domain**	**Wine**
hasWineDescriptor	rdfs:range	WineDescriptor
hasWineDescriptor	rdfs:domain	Wine

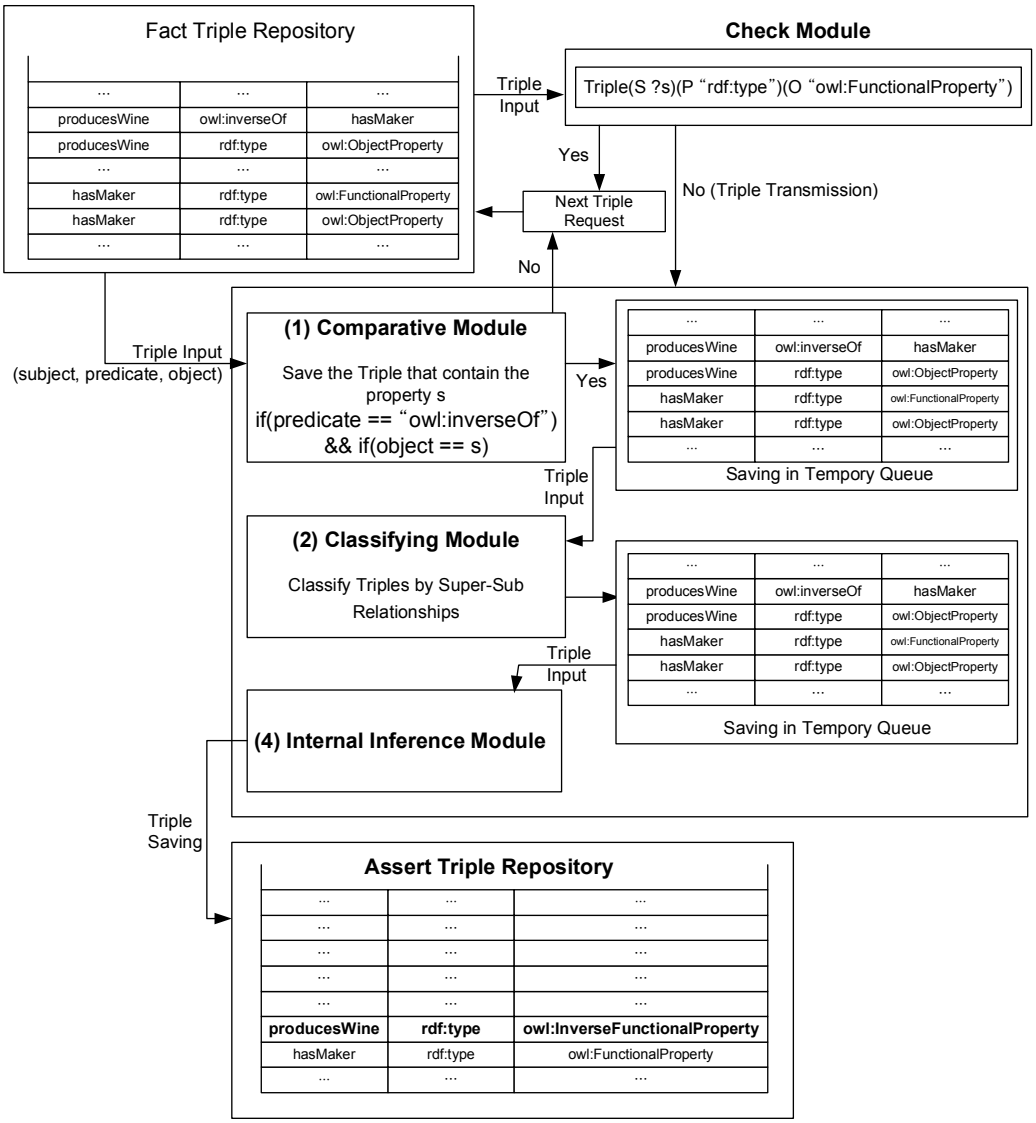
2.5 규칙 5


Fact Triple Repository
MendocinoRegion adjecentRegion SonomaRegion
MendocinoRegion locatedIn CaliforniaRegion
MendocinoRegion rdf:type Region
adjecentRegion rdfs:range Region
adjacentRegion rdfs:domain Region
adjacentRegion rdf:type owl:SymmetricProperty
adjacentRegion rdf:type owl:ObjectProperty
Triple Input
Check Module
Triple(S ?s)(P "rdf:type")(O "owl:SymmetricProperty")
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property s
if(predicate == s)
Yes
MendocinoRegion adjecentRegion SonomaRegion
Saving in Tempory Queue
Triple Input
(2) Classifying Module
Classify Triples by Super-Sub Relationships
MendocinoRegion adjecentRegion SonomaRegion
Saving in Tempory Queue
Triple Input
(3) Serialization Module
Enumeration by Order,
Like (X → P → Y)(Y → P → Z)
Triple Input
(4) Internal Inference Module
MendocinoRegion adjecentRegion SonomaRegion
Saving in Tempory Queue
Triple Saving
Assert Triple Repository
SonomaRegion adjecentRegion MendocinoRegion
MendocinoRegion adjecentRegion SonomaRegion

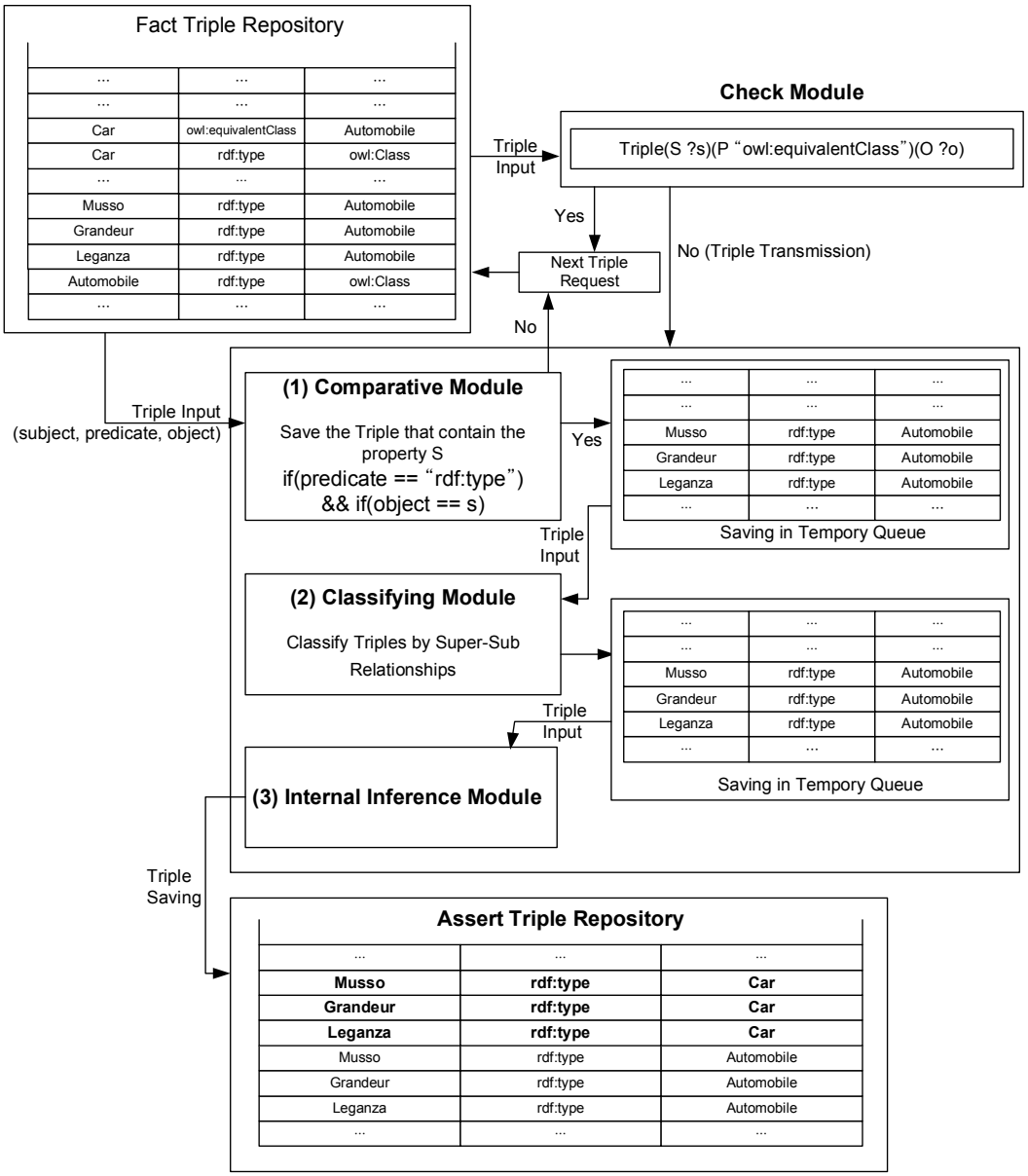
2.6 규칙 6


Fact Triple Repository
...
ChateauDYchemSauterne
madeFromGrape
SauvignonBlancGrape
ChateauDYchemSauterne
madeFromGrape
SemillonGrape
ChateauDYchemSauterne
rdf:type
Sauternes
madeIntoWine
owl:inverseOf
madeFromGrape
madeIntoWine
rdf:type
owl:ObjectProperty
produceWine
owl:inverseOf
hasMaker
produceWine
rdf:type
owl:ObjectProperty
Check Module
Triple(S ?s)(P "owl:inverseOf")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input
(subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property S
If(predicate == s)
Yes
ChateauDYchem Sauterne
madeFromGrape
SauvignonBlanc Grape
ChateauDYchem Sauterne
madeFromGrape
SemillonGrape
Saving in Tempory Queue
Triple Input
(4) Internal Inference Module
Triple Saving
Assert Triple Repository
SauvignonBlanc Grape
madeIntoWine
ChateauDYchemSauterne
ChateauDYchemSauterne
madeFromGrape
SauvignonBlanc Grape
SemillonGrape
madeIntoWine
ChateauDYchemSauterne
ChateauDYchemSauterne
madeFromGrape
SemillonGrape

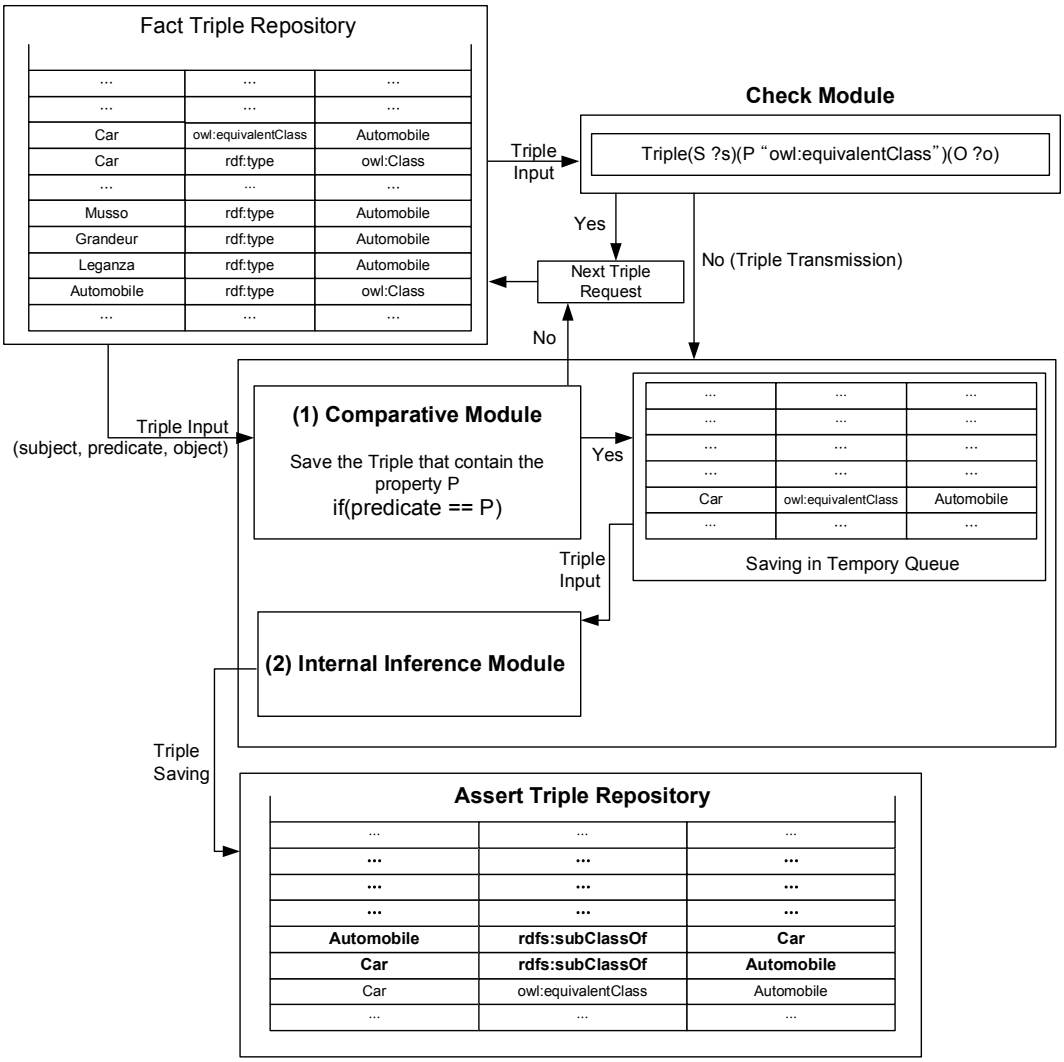
2.7 규칙 7


Fact Triple Repository
Check Module
Triple(S ?s)(P "rdf:type")(O "owl:FunctionalProperty")
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property s
if(predicate == "owl:inverseOf") && if(object == s)
Yes
Saving in Tempory Queue
Triple Input
(2) Classifying Module
Classify Triples by Super-Sub Relationships
Triple Input
Saving in Tempory Queue
(4) Internal Inference Module
Triple Saving
Assert Triple Repository

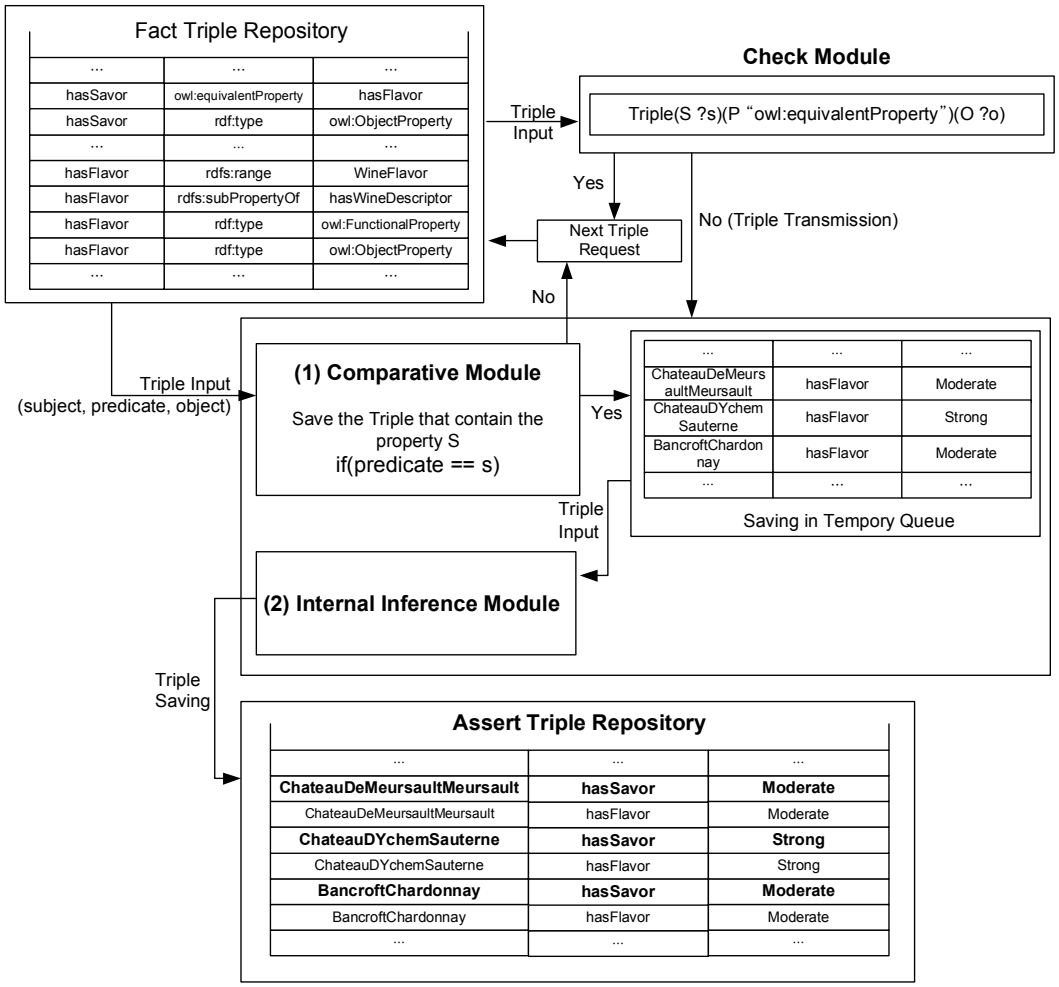
2.8 규칙 8


Fact Triple Repository
Car owl:equivalentClass Automobile
Car rdf:type owl:Class
Musso rdf:type Automobile
Grandeur rdf:type Automobile
Leganza rdf:type Automobile
Automobile rdf:type owl:Class
Check Module
Triple(S ?s)(P "owl:equivalentClass")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property S
if(predicate == "rdf:type") && if(object == s)
Yes
Musso rdf:type Automobile
Grandeur rdf:type Automobile
Leganza rdf:type Automobile
Saving in Tempory Queue
Triple Input
(2) Classifying Module
Classify Triples by Super-Sub Relationships
Musso rdf:type Automobile
Grandeur rdf:type Automobile
Leganza rdf:type Automobile
Saving in Tempory Queue
Triple Input
(3) Internal Inference Module
Triple Saving
Assert Triple Repository
Musso rdf:type Car
Grandeur rdf:type Car
Leganza rdf:type Car
Musso rdf:type Automobile
Grandeur rdf:type Automobile
Leganza rdf:type Automobile

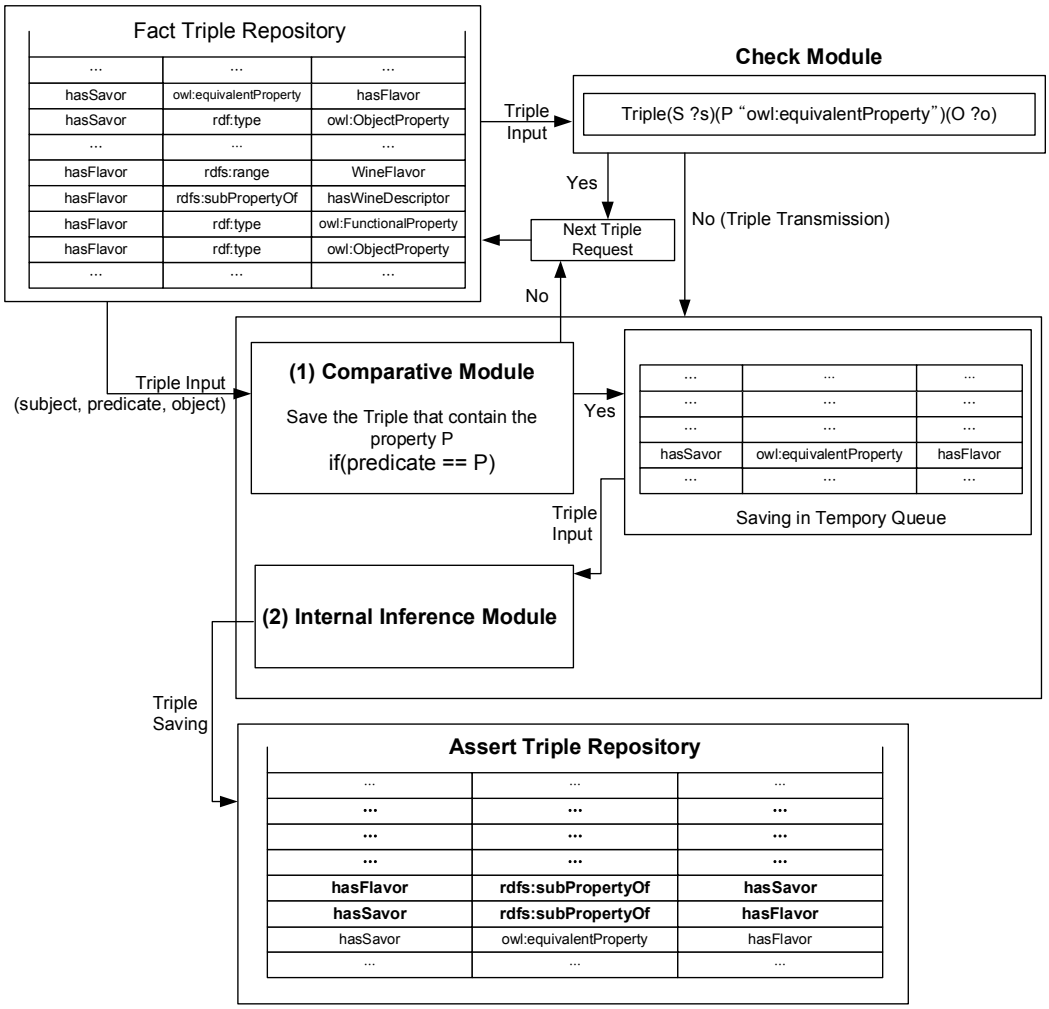
2.9 규칙 9


Fact Triple Repository
Car owl:equivalentClass Automobile
Car rdf:type owl:Class
Musso rdf:type Automobile
Grandeur rdf:type Automobile
Leganza rdf:type Automobile
Automobile rdf:type owl:Class
Check Module
Triple(S ?s)(P "owl:equivalentClass")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property P
if(predicate == P)
Yes
Car owl:equivalentClass Automobile
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
Automobile rdfs:subClassOf Car
Car rdfs:subClassOf Automobile
Car owl:equivalentClass Automobile

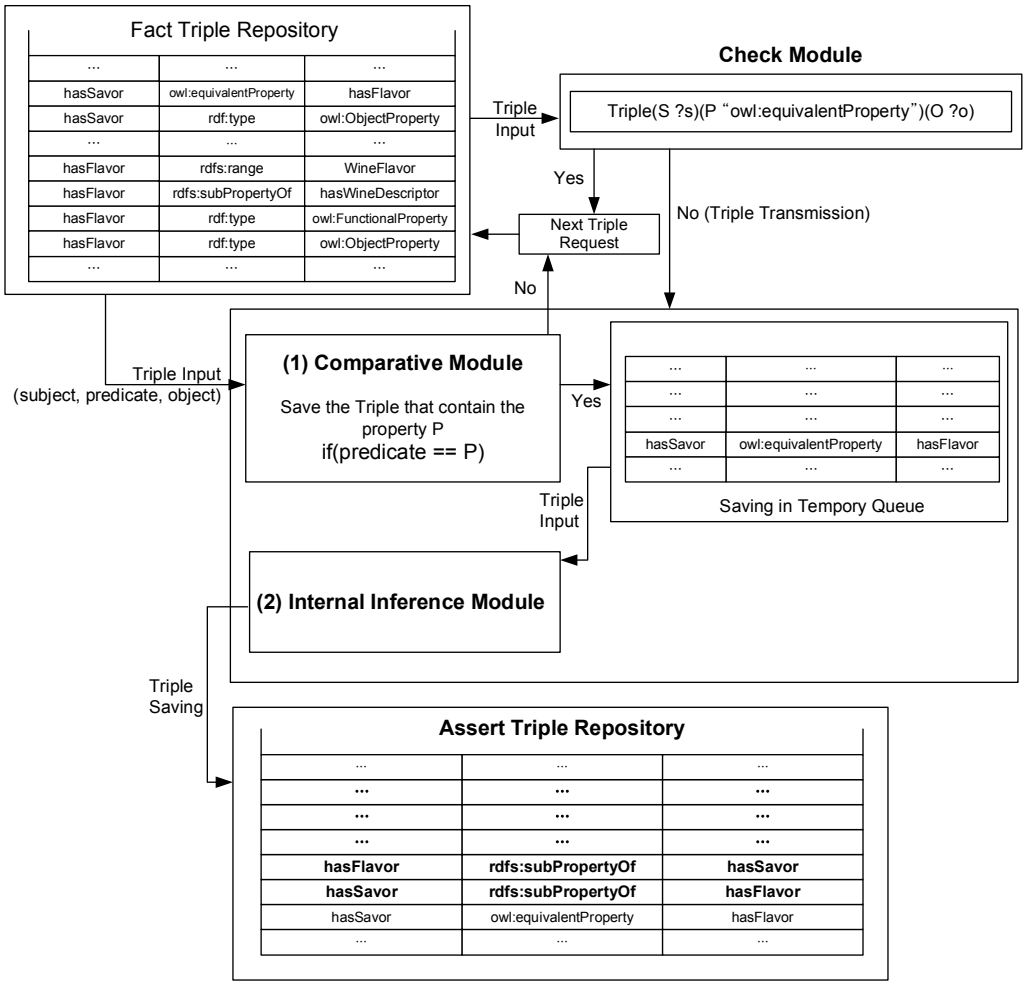
2.10 규칙 10


Fact Triple Repository
...
hasSavor
owl:equivalentProperty
hasFlavor
hasSavor
rdf:type
owl:ObjectProperty
hasFlavor
rdfs:range
WineFlavor
hasFlavor
rdfs:subPropertyOf
hasWineDescriptor
hasFlavor
rdf:type
owl:FunctionalProperty
hasFlavor
rdf:type
owl:ObjectProperty
Check Module
Triple(S ?s)(P "owl:equivalentProperty")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property S
if(predicate == s)
Yes
ChateauDeMeursaultMeursault
hasFlavor
Moderate
ChateauDYchem Sauterne
hasFlavor
Strong
BancroftChardonnay
hasFlavor
Moderate
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
ChateauDeMeursaultMeursault
hasSavor
Moderate
ChateauDeMeursaultMeursault
hasFlavor
Moderate
ChateauDYchemSauterne
hasSavor
Strong
ChateauDYchemSauterne
hasFlavor
Strong
BancroftChardonnay
hasSavor
Moderate
BancroftChardonnay
hasFlavor
Moderate

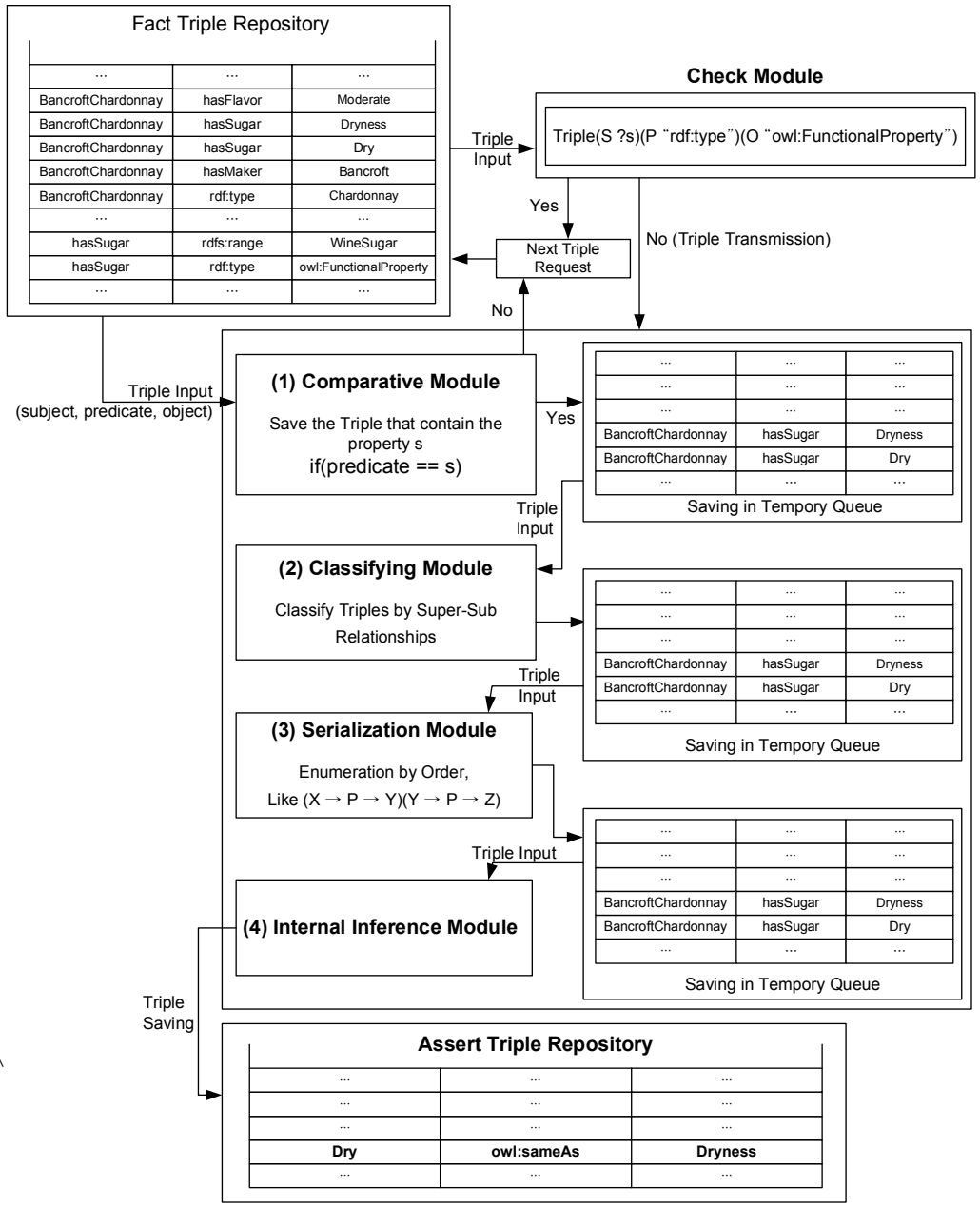
2.11 규칙 11


Fact Triple Repository
...
hasSavor
owl:equivalentProperty
hasFlavor
hasSavor
rdf:type
owl:ObjectProperty
hasFlavor
rdfs:range
WineFlavor
hasFlavor
rdfs:subPropertyOf
hasWineDescriptor
hasFlavor
rdf:type
owl:FunctionalProperty
hasFlavor
rdf:type
owl:ObjectProperty
Triple Input
Check Module
Triple(S ?s)(P "owl:equivalentProperty")(O ?o)
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input
(subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property P
if(predicate == P)
Yes
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
hasFlavor
rdfs:subPropertyOf
hasSavor
hasSavor
rdfs:subPropertyOf
hasFlavor
hasSavor
owl:equivalentProperty
hasFlavor

2.12 규칙 12


Fact Triple Repository
...
hasSavor
owl:equivalentProperty
hasFlavor
hasSavor
rdf:type
owl:ObjectProperty
hasFlavor
rdfs:range
WineFlavor
hasFlavor
rdfs:subPropertyOf
hasWineDescriptor
hasFlavor
rdf:type
owl:FunctionalProperty
hasFlavor
rdf:type
owl:ObjectProperty
Check Module
Triple(S ?s)(P "owl:equivalentProperty")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property P
if(predicate == P)
Yes
hasSavor
owl:equivalentProperty
hasFlavor
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
hasFlavor
rdfs:subPropertyOf
hasSavor
hasSavor
rdfs:subPropertyOf
hasFlavor
hasSavor
owl:equivalentProperty
hasFlavor

2.13 규칙 13

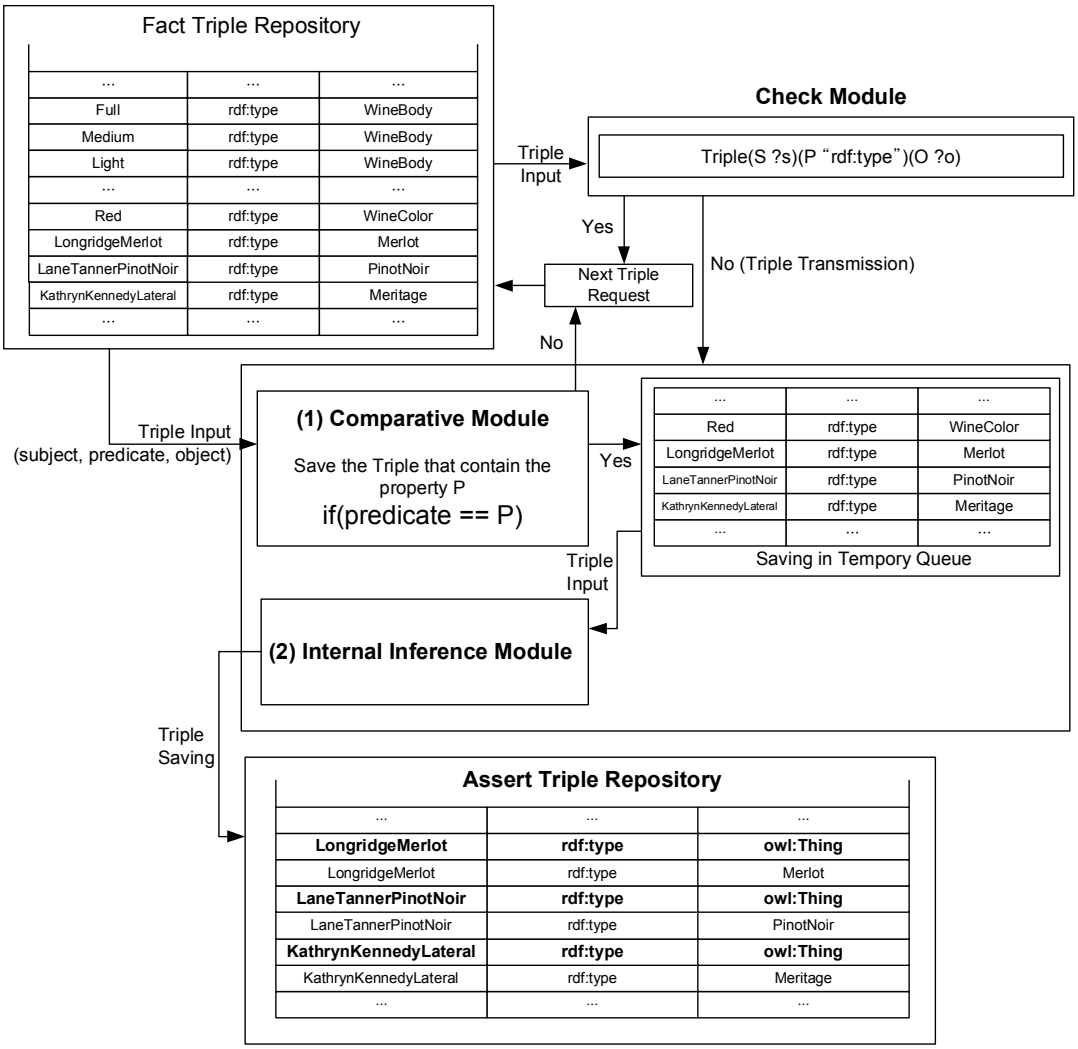


...	...	...
BancroftChardonnay	hasFlavor	Moderate
BancroftChardonnay	hasSugar	Dryness
BancroftChardonnay	hasSugar	Dry
BancroftChardonnay	hasMaker	Bancroft
BancroftChardonnay	rdf:type	Chardonnay
...	...	...
hasSugar	rdfs:range	WineSugar
hasSugar	rdf:type	owl:FunctionalProperty
...	...	...

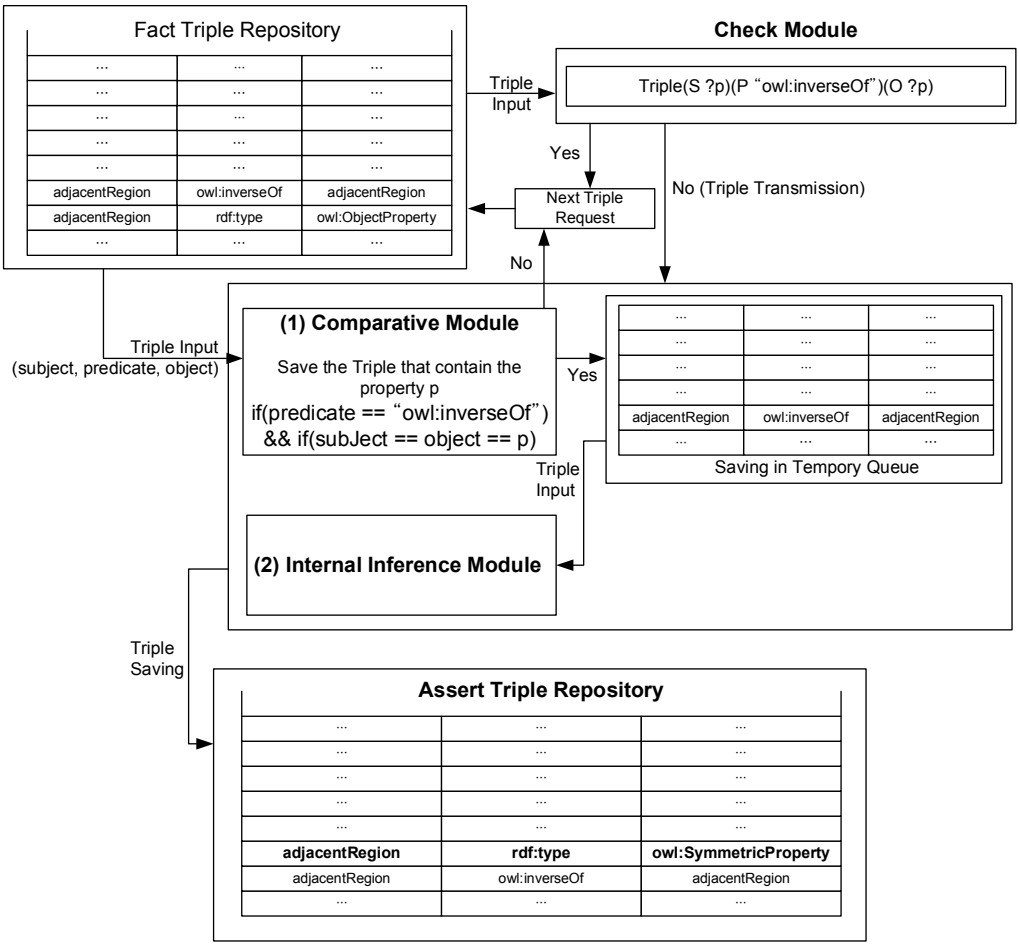
...	...	...
...	...	...
...	...	...
BancroftChardonnay	hasSugar	Dryness
BancroftChardonnay	hasSugar	Dry
...	...	...

...	...	...
...	...	...
...	...	...
Dry	**owl:sameAs**	**Dryness**
...	...	...

2.14 규칙 14


Fact Triple Repository
...
Full
rdf:type
WineBody
Medium
Light
Red
WineColor
LongridgeMerlot
Merlot
LaneTannerPinotNoir
PinotNoir
KathrynKennedyLateral
Meritage
Check Module
Triple(S ?s)(P "rdf:type")(O ?o)
Triple Input
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property P
if(predicate == P)
Yes
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
LongridgeMerlot
rdf:type
owl:Thing
LaneTannerPinotNoir
KathrynKennedyLateral

2.15 규칙 15


Fact Triple Repository
adjacentRegion
owl:inverseOf
adjacentRegion
adjacentRegion
rdf:type
owl:ObjectProperty
Triple Input
Check Module
Triple(S ?p)(P "owl:inverseOf")(O ?p)
Yes
No (Triple Transmission)
Next Triple Request
No
Triple Input (subject, predicate, object)
(1) Comparative Module
Save the Triple that contain the property p
if(predicate == "owl:inverseOf") && if(subJect == object == p)
Yes
adjacentRegion
owl:inverseOf
adjacentRegion
Saving in Tempory Queue
Triple Input
(2) Internal Inference Module
Triple Saving
Assert Triple Repository
adjacentRegion
rdf:type
owl:SymmetricProperty
adjacentRegion
owl:inverseOf
adjacentRegion

저작물 이용 허락서

학 과	전자계산학과	학 번	20047339	과 정	박사
성 명	한글 공 현 장	한문 孔 賢 章	영문 KONG HYUN JANG		
주 소	광주광역시 북구 동림동 1175 동천마을 2단지 208동 1004호				
연락처	E-mail : kisofire@chosun.ac.kr				
논문 제목	[한글] 의미적 정보검색을 위한 온톨로지 자동 구축 및 효율적 질의 처리 방법				
	[영문] The Method of Automatic Ontology Building and Efficient Query Processing for the Semantic Information Retrieval				

본인이 저작한 위의 저작물에 대하여 다음과 같은 조건 아래 조선대학교가 저작물을 이용할 수 있도록 허락하고 동의합니다.

- 다 음 -

1. 저작물의 DB구축 및 인터넷을 포함한 정보통신망에의 공개를 위한 저작물의 복제, 기억장치에의 저장, 전송 등을 허락함.
2. 위의 목적을 위하여 필요한 범위 내에서의 편집과 형식상의 변경을 허락함. 다만, 저작물의 내용변경은 금지함.
3. 배포・전송된 저작물의 영리적 목적을 위한 복제, 저장, 전송 등은 금지함.
4. 저작물에 대한 이용기간은 5년으로 하고, 기간종료 3개월 이내에 별도의 의사 표시가 없을 경우에는 저작물의 이용기간을 계속 연장함.
5. 해당 저작물의 저작권을 타인에게 양도하거나 출판을 허락을 하였을 경우에는 1개월 이내에 대학에 이를 통보함.
6. 조선대학교는 저작물 이용의 허락 이후 해당 저작물로 인하여 발생하는 타인에 의한 권리 침해에 대하여 일체의 법적 책임을 지지 않음.
7. 소속 대학의 협정기관에 저작물의 제공 및 인터넷 등 정보통신망을 이용한 저작물의 전송・출력을 허락함.

동의여부 : 동의( ) 반대( )

2007 년 8 월 일

저작자: 공 현 장 (인)

조선대학교 총장 귀하