논문 2010-47CI-4-12

# 온톨로지 파싱 속도향상을 위한 온톨로지 파서 설계

## ( Ontology Parser Design for Speed Improvement of Ontology Parsing )

김 원 필*, 공 현 장**

( Won-Pil Kim and Hyun-Jang Kong )

요 약

시맨틱 웹에서 핵심 연구는 온톨로지 파싱의 효율성이다. 온톨로지의 효율적 파싱과 추론은 시맨틱 웹의 궁극적인 목적인 의미적인 정보검색의 기반이 된다. 그러나 기존의 온톨로지 저작도구들은 온톨로지 파싱속도에 있어 효율적이지 못하고 있는 실정이다. 따라서, 본 연구에서는 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 추출한다. 이에 본 연구에서 설계한 파서의 속도는 기존의 도구들의 파서보다 빠른 처리가 이루어짐을 확인할 수 있었다.

Abstract

The core study of semantic web is the efficiency of ontology parsing. The ontology parsing and inference is based on the significant information retrieval which is the ultimate purpose of semantic web. However, most existing ontology writing tools were not processing the efficient ontology parsing. Therefore, we design the two steps ontology parser for extracting the all facts, are included in the ontology, more fast in this study. In the first step, the token extractor collects the all tokens of ontology and the triple extractor extracts the statements in the collected tokens. In conclusion, we confirm that which is designed in this study, processes the ontology parsing more faster than the existing ontology parsers.

**Keywords**: Ontology Parser, Ontology Editor, Jena, JESS

## I. 서 론

시맨틱 웹에서 핵심 연구는 온톨로지 파싱의 효율성이다. 온톨로지의 효율적 파싱과 추론은 시맨틱 웹의 궁극적인 목적인 의미적인 정보검색의 기반이 된다. 그러나 기존의 KAON, Protege 3.0, OilEd, OntoEdit, OWLeditor[1]등의 온톨로지 저작도구들은 온톨로지 파싱속도에 있어 효율적이지 못하고 있는 실정이다. 시맨틱 웹에서 의미적 정보검색의 효율성을 위해 파싱의 속도 향상은 매우 중요한 요소이다. 따라서, 본 연구에서는 OWL Full 문서를 효율적으로 파싱하는 범용적인 파싱 방법을 제안하고 설계하였다.

HP Labs Semantic Web research group에서 연구중인 Jena[2]는 시맨틱 웹을 위한 어플리케이션들을 위해 작성된 Java 프레임웍이다. Jena는 RDF(S), OWL과 DAML+OIL의 파싱을 위한 프로그램 환경을 제공한다. Jena는 기존의 추론 엔진들 중 가장 활발히 연구되고 있으며, 시맨틱 웹 실현에 기여도가 상당히 높다. 하지만, Jena는 OWL문서의 파싱에 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.

* 정회원, 조선이공대학교 사이버보안과
(Chosun university college of science & technology)
** 정회원, 조선대학교병원 전산실
(chosun university hospital)
접수일자: 2010년2월10일, 수정완료일: 2010년7월7일

Hoolet[3]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[4~5]는 미국의 Carnegie Mellon 대학에서 개발한 OWL 추론 엔진은 가장 체계적이고 완성도 높은 추론 엔진이라 할 수 있다. Jena와 F-OWL[6]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 파싱을 위해 JESS의 추론 모듈을 채택하여 추론 규칙과 내용을 분석한 후, 온톨로지 파싱 방법에 적용하였다.

## II. 온톨로지 파서 설계

JESS는 OWL 어휘들이 갖고 있는 고유관계들 표 1과 같이 30가지와 각 어휘에 적용될 추론 규칙 20가지를 정의하고 있다.

표 1에서 보는바와 같이 어휘들 사이의 고유관계를 정의하였으며, 어휘들 간의 관계와 어휘가 갖고 있는 고유 Axiom들을 이용하여 표 2의 20가지 추론 규칙을 정의하고 있다.

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 Jess에서 정의된 30가지 OWL의 구조적 규칙에 의한 온톨로지가 포함하고 있는 Fact 트리플을 추출하는 1차 파싱과 이렇게 파싱된 Fact 트리플에서 Jess에서 정의된 20가지 추론 규칙을 적용한 2차 파싱을 의미한다. 본 연구에서 온톨로지가 기술하는 모든 사실을 빠르게 추출하기 위해 2단계에 걸친 온톨로지 파서를 설계하였다. 정확하고 빠른

표 1. OWL 어휘들의 고유관계
Table 1. Inherent relation of Vocabulary.

| 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 |

표 2. JESS의 추론규칙
Table 2. Inference rule of JESS.

| | 규칙 이름 | 추론 방식 |
|---|---|---|
| 1 | Propagate Transitive Properties | Triple(S ?p)(P rdf:type)(O owl:TransitiveProperty)<br>Triple(S ?x)(P ?p)(O ?y)<br>Triple(S ?y)(P ?p)(O ?z)<br>=> 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)<br>Triple(S ?x)(P rdf:type)(O ?y)<br>=> 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)<br>Triple(S ?a)(P ?y)(O ?b)<br>=> Assert Triple(S ?a)(P ?z)(O ?b) |
| 4 | Default behavior for signature propagation | Triple(S ?x)(P rdfs:subPropertyOf)(O ?y)<br>Triple(S ?y)(P rdfs:domain)(O ?z)<br>=> Assert Triple(S ?x)(P rdfs:domain)(O ?z) |
| | | Triple(S ?x)(P rdfs:subPropertyOf)(O ?y)<br>Triple(S ?y)(P rdfs:range)(O ?z)<br>=> 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)<br>Triple(S ?x)(P ?p)(O ?y)<br>=> Assert Triple(S ?y)(P ?p)(O ?x) |
| 6 | Complete with instances of inverse properties | Triple(S ?p1)(P owl:inverseOf)(O ?p2)<br>Triple(S ?x)(P ?p1)(O ?y)<br>=> 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)<br>Triple(S ?p1)(P rdf:type)(O owl:FunctionalProperty)<br>=> 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)<br>Triple(S ?p1)(P rdf:type)(O owl:inverseFunctionalProperty)<br>=> 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)<br>Triple(S ?x)(P rdf:type)(O ?t1)<br>=> 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)<br>=> Assert Triple(S ?t1)(P rdfs:subClassOf)(O ?t2)<br>=> 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)<br>Triple(S ?o1)(P rdf:type)(O ?c1)<br>Triple(S ?o2)(P rdf:type)(O ?c2)<br>=> 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)<br>Triple(S ?c2)(P rdfs:subClassOf)(O ?c1)<br>Test(nnq ?c1 ?c2)<br>=> 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)<br>Triple(S ?s1)(P ?p)(O ?o)<br>=> 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)<br>Triple(S ?s)(P ?p1)(O ?o)<br>=> 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)<br>=> Assert Triple(S ?p1)(P rdfs:subPropertyOf)(O ?p2)<br>=> 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)<br>Triple(S ?p2)(P rdfs:subPropertyOf)(O ?p1)<br>Test(neq ?p1 ?p2)<br>=> 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)<br>Triple(S ?s)(P ?p)(O ?o1)<br>Triple(S ?s)(P ?p)(O ?o2)<br>Test(neq ?o1 ?o2)<br>=> 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)<br>Triple(S ?s1)(P ?p)(O ?o)<br>Triple(S ?s2)(P ?p)(O ?o)<br>Test(neq ?s1 ?s2)<br>=> 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)<br>=> 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)<br>Assert Triple(S ?p)(P rdf:type)(O owl:SymmetricProperty) |

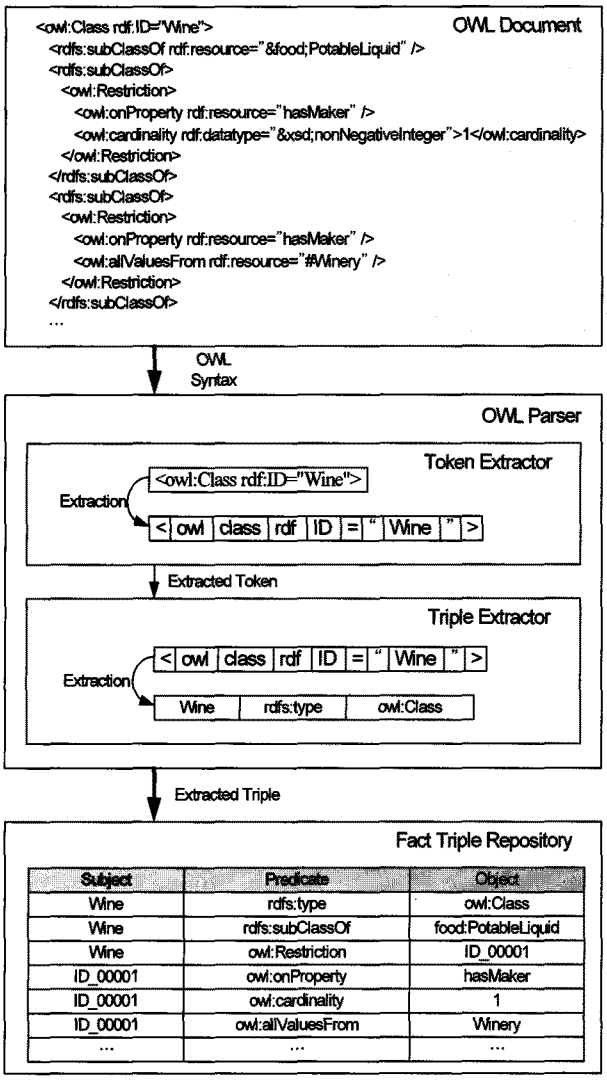



그림 1. 파서 처리 과정
Fig. 1. Process of Parser.

파서의 설계를 위해 토큰 추출기에서 온톨로지의 구문의 토큰을 1단계로 추출하고, 이를 바탕으로 트리플 추출기에서 Statement를 추출한다. 본 연구에서 제안하는 추론 규칙 기반의 OWL 온톨로지 파서 설계도는 그림 1과 같다.

온톨로지 구문의 파싱을 위해 작성된 OWL 온톨로지를 입력한다. 온톨로지 파서 내의 토큰 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 저장된다. OWL은 사용의 목적에 따라 Lite, DL과 Full

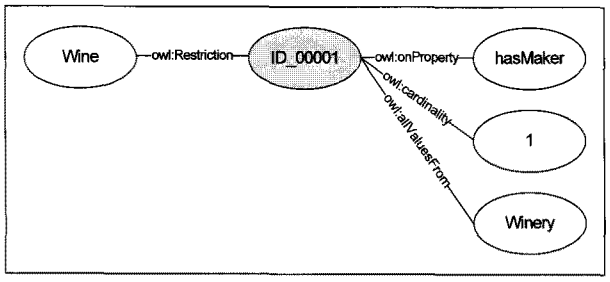



그림 2. Blank Node 처리
Fig. 2. Process of Blank Node.

의 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. 그림 2에서 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 2와 같이 특정의 ID를 이용하여 Blank Node로 처리하여 속성이 갖는 지역 제약의 내용을 모두 기술하도록 설계하였다.

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.

## III. 실 험

표 3은 기존의 온톨로지 저작 도구들과 비교한 내용이다.

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 4, 5와 같다.

표 4의 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

표 3. 온톨로지 저작 도구 비교
Table 3. Comparison of Ontology editor.

| 저작 도구 | 편집 방식 | 파일 형태 | 자동 구축 | 추론기능 | 특징 | 파싱 속도 |
|---|---|---|---|---|---|---|
| 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 |
| 제안된 저작 도구 | Tree & Form | OWL | 지원 | JESS 추론지원 | 트리플 기반 질의처리 | 빠름 |

표 4. Jena의 RDF 온톨로지 파싱 모듈
Table 4. RDF Ontology parsing module of Jena.

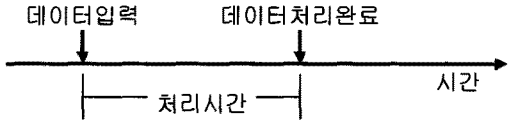
```
// list the statements in the graph
StmtIterator iter = model.listStatements();
// print out the predicate, subject and object of each
statement while (iter.hasNext()) {
Statement stmt  = iter.next();  // get next statement
Resource  subject = stmt.getSubject(); // get the subject
Property  predicate = 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 5. 본 연구에서 제안한 온톨로지 파싱 모듈
Table 5. Parsing module of Ontology.

```
// 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(" .");
}
```

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. 표 5는 본 연구에서 제안한 온톨로지 RDF, OWL 파싱 모듈이다.

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 6과 같다.

표 6의 파서 처리 시간에 근거하여 각 저작도구 파서의 단위 시간당 트리플 처리량(throughput)을 측정하였으며, 그 기준은 단위시간(1초)동안 트리플 생성량을 비교하였다.

표 6. 온톨로지 저작 도구의 파싱 속도
Table 6. Parsing velocity of Ontology editor.

(단위: 초)

| 저작도구 | KAON | Protege 3.0 | OilEd | OWL Editor | 제안된 저작도구 |
|---|---|---|---|---|---|
| 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) |

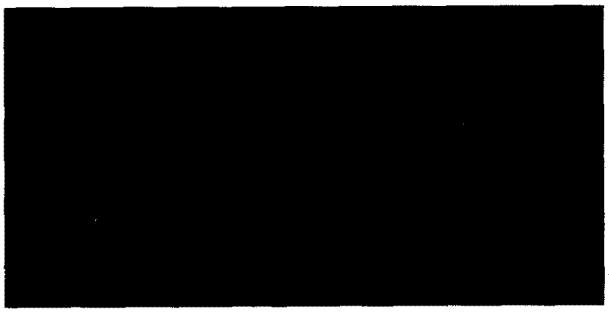


그림 3. 온톨로지 저작 도구의 단위 시간당 트리플 처리량
Fig. 3. Triple process quantity per hour of Ontology editor.

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 6에서 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 전체 파싱의 속도에 영향을 미치지 않을 만큼 빠르게 파싱한다.

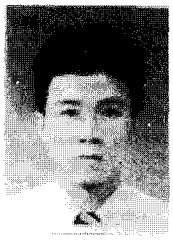
## IV. 결 론

시맨틱웹에서 의미적 정보검색을 위해 온톨로지 파싱의 속도는 중요한 요소이다. 그러나 기존의 온톨로지 저작도구들은 온톨로지 파싱속도에 있어 효율적이지 못하고 있는 실정이다. 따라서 본 연구에서는 온톨로지가 기술하는 모든 사실을 빠르고 효율적으로 추출하기 위해 2단계에 걸쳐 OWL Full 문서를 효율적으로 파싱하는 범용적인 파싱방법을 제안하고 설계하였다. 이에 본 연구에서 설계한 파서의 속도는 기존의 도구들의 파서보다 빠른 처리가 이루어짐을 확인할 수 있었다.

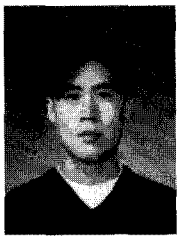
## 참 고 문 헌

[1] Michael Denny, "Ontology editor survey results", http://xml.com/2002/11/06/Ontology_Editor_Survey.html/, (2002).
[2] "Jena 2 - A Semantic Web Framework", http://www.hpl.hp.com/semweb/jena.htm
[3] "OWL: Hoolet", http://owl.man.ac.uk/hoolet/.
[4] Fabien L. Gandon, Norman M. Sadeh, "OWL Inference Engine in Jess", http://www.cs.cmu.edu/~sadeh/MyCampusMirror/OWLEngine.html.
[5] "Jess, the Rule Engine for the Java Platform", http://herzberg.ca.sandia.gov/jess/References.
[6] "F-OWL: An OWL Inference Engine in Flora-2", http://fowl.sourceforge.net/about.html/

## 저 자 소 개



김 원 필(정회원)
1994년 호남대학교 전산통계학과 학사 졸업.
1999년 호남대학교 컴퓨터공학과 석사 졸업.
2004년 조선대학교 전자계산학과 박사 졸업.
2008년~현재 조선이공대학 사이버보안과교수
<주관심분야 : 유비쿼터스, 정보검색, 정보보안>



공 현 장(정회원)
2002년 조선대학교 전자계산학과 학사 졸업.
2004년 조선대학교 전자계산학과 석사 졸업.
2007년 조선대학교 전자계산학과 박사졸업
2008년~현재 조선대학교 병원 전산실
<주관심분야 : 유비쿼터스, 정보검색, 정보보안>