EOL Reasoner : 온톨로지 기반 지식 추론 엔진

EOL Reasoner : Ontology-based knowledge reasoning engine

전형백, Hyeongbaek Jeon*, 이건수, Keonsoo Lee**, 김민구, Minkoo Kim***

요약 유비쿼터스 컴퓨팅 환경에서는 사용자의 다양한 요구에 부응하기 위하여 지식을 활용한 서비스의 필요성이 증가하고 있다. 온톨로지는 이같은 분산 컴퓨팅 오브젝트들 사이에서 지식 공유를 위한 표현 방법으로 많이 사용되고 있다. 이러한 온톨로지를 표현하고 처리하기 위해 본 연구에서는 Description Logic 기반 지식 표현 언어인 EOL을 바탕으로 지식을 활용하기 위한 추론 엔진을 구현하였다. EOL 은 간결하고 직관적인 문법으로 비전문가도 EOL을 사용해 지식을 쉽게 표현 및 활용할 수 있다. 본 논문에서는 이러한 EOL 로 표현된 지식을 보다 효율적으로 처리하기 위해 지식의 표현 범위에 따라 각기 다른 추론 알고리즘을 사용하는 추론 엔진을 구현하였다.

Abstract These days, computing systems need to be intelligent for satisfying general users' ambiguous requests. In order to make a system intelligent, several methods of managing knowledge have been proposed. Especially, in ubiquitous computing environment, where various computing objects are working together for achieving the given goal, ontology can be the best solutionfor knowledge management. In this paper, we proposed a novel reasoner processing ontology-based knowledge which is expressed in EOL. As this EOL reasoner uses less computing resource, it can be easily adapted to various computing objects in ubiquitous computing environment providing easy usability of knowledge.

핵심어: Knowledge, Ubiquitious Computing, Description Logic

본 논문은 2007년 아주대학교 학술 연구비 지원에 의하여 연구되었음.

*주저자 : 아주대학교 정보통신전문대학원 e-mail: emgau@ajou.ac.kr

**공동저자 : 아주대학교 정보통신전문대학원 e-mail: lks7256@ajou.ac.kr

***교신저자 : 아주대학교 정보 및 컴퓨터공학부 교수; e-mail: minkoo@ajou.ac.kr

1. 서론

유비쿼터스 컴퓨팅 환경이란, 해당 공간의 모든 컴퓨팅 오브젝트들이 사용자의 명시적 요청 없이도 서비스를 제공할 수 있는 환경을 의미한다.[1] 이러한 서비스를 제공하기 위해 공간 내의 컴퓨팅 오브젝트들은 기능의 수행 여부를 판단할 수 있어야 하는데, 이를 위해서는 지식이 필요해진다. 유비쿼터스 컴퓨팅 환경에서 컴퓨팅 오브젝트들은 네트워크를 통해 여러 가지 지식을 공유할 수 있고 이를 통해서 협력하기도 한다. 이때, 도메인에 따라 지식의 양도 다르고 그 활용 방법도 다양해진다. 또한, 매우 다양한 능력을 가진 컴퓨팅 오브젝트가 존재할 수 있기 때문에, 지식을 중앙에 집중시키는 방식보다는 지식을 각 컴퓨팅 오브젝트들에게 분산시키고, 이를 서로 공유함으로써 서비스를 제공하는 방식이 필요해진다.

컴퓨팅 오브젝트 사이에 공유를 가능케 하는 지식 표현의 대표적인 예로 온톨로지가 있다. 본 연구에서는 컴퓨팅 오브젝트들의 온톨로지 지식을 description logic 기반인 EOL 을 이용해 표현하였고, 이를 처리할 수 있는 추론 엔진을 구현하였다.[2][3]

2. 본론

2.1 EOL(Epistemological Ontology Language)

온톨로지를 표현하기 위해 사용되는 언어로는 W3C 에서 제안된 OWL 과 RDF-S 등 여러 가지가 있다.[4] 이들은 XML 기반의 마크업 언어이기 때문에, 온톨로지를 표현하려면 기본 지식이 필요하거나 에디터의 도움이 필요하다. EOL 은 사용자 직관적인 명령어와 표현 문법을 제공함으로써, 추가적인 에디터의 도움 없이도 지식을 표현할 수 있고, 비전문가도 쉽게 활용할 수 있다는 장점이 있다.[2]

Description Logic 으로써 SUNHI 의 표현 범위를 갖는 기존 EOL 은 7개의 정의문(define command), 1개의 삽입문(assert command), 8개의 질의문(query command)을 제공하는데, 여기에는 롤 인스턴스(role instance) 가 고려되지 못했다는 단점이 있었다.본 연구에서는 이를 개선하여 롤 인스턴스에 대한 명령어들을 추가하였다. EOL 추론엔진은 〈표 1〉과 같은 명령어들을 처리한다.

2.2 EOL Reasoner

2.2.1 EOL Reasoner 구조

EOL Reasoner의 구조는 〈그림 1〉과 같다. 크게 User Interface 와 Parser, Knowledgebase, Processor, Reasoner 로 구성되어 있는데, Knowledgebase 는 다시 TBox 와 ABox 로 나뉜다. TBox는 임의의 최상위 노드를 기준으로 컨셉(Concept) 및 롤(role)을 노드로, 노드 간의 포함 관계를 edge 로 하는 acyclic directed graph 를 컨셉과 롤에 대해 각각 하나씩 가진다. 한편, ABox 는 TBox 에 저장되어 있는 컨셉 혹은 롤에 대한 인스턴스(instance)들을 저장한다.

표 1 Commands of EOL Reasoner

Commands for defining concept	
Define concept	dconcept ⟨concept_name⟩ ⟨concept⟩
Define subsume concept	dsubconcept ⟨concept_name⟩ of ⟨concept_name⟩
Define disjoint concept	ddisjointconcept ⟨concept_name⟩ ⟨concept_name⟩
Commands for defining role	
Define role	drole ⟨role_name⟩.⟨concept⟩ ⟨concept_name⟩
Define subsume role	dsubrole ⟨role_name⟩ of ⟨role_name⟩
Define transitive role	dtransitiverole ⟨role_name⟩
Define inverse role	dinverserole ⟨role_name⟩ ⟨role_name⟩
Commands for asserting instance	
Assert concept instance	aconceptinstance ⟨concept_name⟩ ⟨instance_name⟩
Assert role instance	aroleinstance ⟨role_name⟩.⟨instance_name⟩ ⟨instance_name⟩
Commands for querying	
Check satisfiability	issatisfiable ⟨concept⟩
Check equivalence	isequivalent ⟨concept_name⟩ ⟨concept_name⟩
Check subsumption	issubsume ⟨concept_name⟩ of ⟨concept_name⟩
Check disjointness	isdisjoint ⟨concept_name⟩ ⟨concept_name⟩
Retrieve concept	rconcept ⟨concept_name⟩
Retrieve role	rrole ⟨role_name⟩
Retrieve superconcept	rsuperconcept ⟨concept_name⟩
Retrieve subconcept	rsubconcept ⟨concept_name⟩
Retrieve superrole	rsuperrole ⟨role_name⟩
Retrieve subrole	rsubrole ⟨role_name⟩
Retrieve concept instance	rconceptinstance ⟨concept⟩
Retrieve role instance	rroleinstance ⟨role_name⟩

User Interface 를 통해 입력된 명령어는 먼저 Asserting Knowledge 와 Querying Knowledge 로 나뉘는데, Knowledgebase 에 새로운 지식을 추가하는 명령어는 Asserting Knowledge 로, Knowledgebase 에 저장된 지식에 대해 질의를 하는 명령어는 Querying Knowledge 로 구분된다.

Asserting Knowledge 로 구분된 명령어는 임시로 Candidate Knowledgebase 에 저장되고, Processor 는 Reasoner 를 이용하여 의해 입력된 지식에 대해 Satisfiability Check 를 수행한다. Satisfiable 한 지식의 경우, Refined Knowledgebase 에 저장되고, 그렇지 않은 경우 입력이 실패한다. Querying Knowledge 로 구분된 명령어는 다시

Processor 로 넘겨진다. Processor 는 Refined Knowledgebase 를 검색해서 조건에 맞는 지식을 찾아 User Interface 로 출력한다.

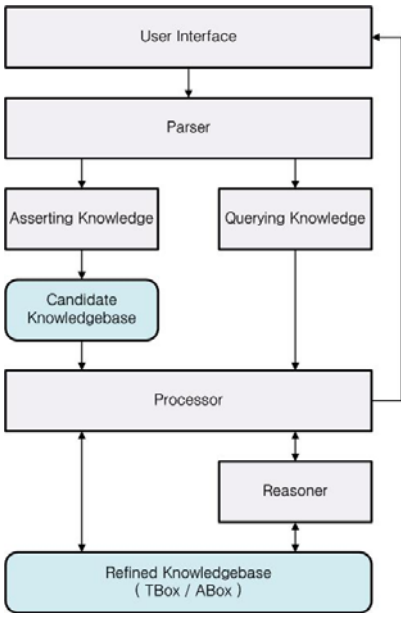

그림 1 EOL 추론 엔진의 구조

2.2.2 추론

Description Logic System 이 제공해야 하는 추론 기능은 크게 Concept Satisfiability, Subsumption, Consistency, Instance Checking 네 가지가 있다.[5] 이 중 Consistency 와 Instance Checking 은 Concept Satisfiability 와 Subsumption 을 이용해 해결할 수 있다. Concept Satisfiability 와 Subsumption 을 판정하기 위한 알고리즘으로 Structural Subsumption 알고리즘과 Tableau-based 알고리즘이 있다.

Structural Subsumption 알고리즘은 문법 기반 추론 방식으로 두 컨셉 C, D 간의 포함 관계를 판정해 내는 알고리즘으로, 시간복잡도는 $O(|C| \times |D|)$ 이다.[6]

다만, Structural Subsumption 알고리즘은 몇 가지 단점을 가지고 있는데, 첫째로 추론 방법의 soundness 를 증명하기는 쉬운 반면에, complete 하지 않다. 둘째, Structural Subsumption 알고리즘을 적용할 수 있는 Description Logic 의 표현 범위에 제한이 있으며, ALN 의 표현 범위까지 적용할 수 있다. 즉, Structural Subsumption 알고리즘만으로는 EOL 로 표현된 지식을 모두 처리할 수 없다.

Tableau-based 알고리즘은 초기 조건을 이용해 tableau 를 구성한 후, 룰에 따라 이를 확장해 나가다가 모순(clash)이 발생할 경우 unsatisfiable 한 것으로, 더 이상 확장할 수 없을 때까지 모순이 발생하지 않으면(open) satisfiable 한 것으로 판정하는 알고리즘이다. Tableau-based 알고리즘의 장점은 logic 의 표현 범위가 커지면 그에 따라 적합한 룰을 추가하여 적용할 수 있다. 다만 표현 범위와 그에 따른 추론 복잡도의 trade-off 는 그대로 작용한다.

EOL 추론 엔진은 경우에 따라 각각 위 두 가지 알고리즘을 사용하는데, 지식의 표현 범위에 따라 좀 더 적합한 성능을 얻을 수 있는 추론 방법을 택한다. 기본적으로 Tableau-based 알고리즘을 사용하며, 추론에 필요한 지식의 표현 범위가 ALN 이하일 경우, Structural Subsumption 알고리즘을 사용한다.[7][8]

표 2 Transformation rules

and – rule	
Condition	L contains (and C D)(x), but not both C(x) and D(x).
Action	L' := L ∪{ C(x), D(x) }.
or – rule	
Condition	L contains (or C D)(x), but neither C(x) nor D(x).
Action	L' := L(x) ∪ { C(x) }, L'' := L(x) ∪ { D(x) }.
some – rule	
Condition	L contains (some role.C)(x), but there is no individual name y such that C(y) and role(x, y) are in L.
Action	L' := L ∪ { C(y), role(x, y) } where y is an individual name not occurring in L.
all –rule	
Condition	L contains (all role.C)(x) and role(x, y), but it does not contain C(y).
Action	L' := L ∪ { C(y) }.
atleast – rule	
Condition	L contains (atleast n role)(x), and there are no individual $y_1, \cdots, y_n$ such that role(x, y_i) ($1 \le i \le n$) and $y_i \neq y_j$ ($1 \le i \le j \le n$) are in L.
Action	L' := L ∪ { role(x, y_i) | $1 \le i \le n$ } ∪ { $y_i \neq y_j$ | $1 \le i \le j \le n$ }, where $y_1, \cdots, y_n$ are distinct individual names not occurring in L.
atmost – rule	
Condition	L contains distinct individual names $y_1, \cdots, y_{n+1}$ such that (atmost n role)(x) and role(x, y_1), ⋯, role(x, y_{n+1}) are in L, and $y_i \neq y_j$ is not in L for some i, j, $1 \le i \le j \le n+1$.
Action	For each pair y_i, y_j such that $1 \le i \le j \le n+1$ and $y_i \neq y_j$ is not in L, the labeled ABox $L_{i,j} := [y_i / y_j]L$ is obtained from L by replacing each occurrence of y_i by y_j.

Tableau-based 알고리즘에 따라 Satisfiability check 를 하기 위해서는, 각 constructor 에 대응하는 transformation rule 들이 필요하다. EOL 추론 엔진에 적용되는 Tableau-based 알고리즘의 Transformation rule 은 〈표 2〉와 같다. C 와 D 는 임의의 컨셉을 의미하고 role 은 임의의 롤을 의미하며, x, y 는 인스턴스를, L 은 Labeled ABox 를 의미한다. 여기서 Labeled ABox 란, ABox 에 포함되어 있는 인스턴스가 가지고 있는 속성들을 원소로 하는 집합을 의미한다. 즉, ABox 가 x 라는 인스턴스를 포함할 때, x 의 컨셉들과 x 의 롤들이 Labeled ABox 의 원소가 된다. 해당 지식의 각 constructor 에 따라 Labeled ABox 가 변화되는데, 최종적으로 더 이상 적용할 transformation rule 이 없으면서 Labeled ABox 내의 지식들 간에 모순이 일어나지 않으면 satisfiable 한 것으로, transformation rule 을 적용해 나가다

가 모순이 발생하면 unsatisfiable 한 것으로 판단한다. 한편, or-rule 의 경우에는 Labeled ABox 가 2개로 분기하면서 transformation rule 을 계속 적용해 나가게 되는데, 이러한 경우 분기한 Labeled ABox 중 하나라도 satisfiable 한 것으로 판단되면 해당 지식은 satisfiable 한 것으로, 분기한 Labeled ABox 가 모두 unsatisfiable 하면 해당 지식 역시 unsatisfiable 한 것으로 판단한다.

표 3 포함 관계 판정 과정 예

C : (and (some R.A) (some R.B) D : (some R.(and A B))

1. (and C (not D))
L = { (and (some R.A) (some R.B) (not (some R.(and A B))))(x) }
2. Normalization
L = { (and (some R.A) (some R.B) (all R.(or (not A) (not B))))(x) }
3. and-rule 적용 : (and (some R.A) (some R.B) (all R.(or (not A) (not B))))(x)
L = L ∪ { (some R.A)(x), (some R.B)(x), (all R.(or (not A) (not B)))(x) }
4. some-rule 적용 : (some R.A)(x), (some R.B)(x)
L = L ∪ { A(y), R(x,y), B(z), R(x,z) }
5. all-rule 적용 : (all R.(or (not A) (not B)))(x)
L = L ∪ { (or (not A) (not B))(y), (or (not A) (not B))(z) }
6 or-rule 적용 : (or (not A) (not B))(y)
L1 = L ∪ { (not A)(y) } : **모순**
L2 = L ∪ { (not B)(y) }
7. or-rule 적용 : (or (not A) (not B))(z)
L3 = L2 ∪ { (not A)(z) }
L4 = L2 ∪ { (not B)(z) } : **모순**
8. Therefore, (and C (not D)) is **satisfiable.**

1. (and (not C) D)
L = { (and (not (and (some R.A) (some R.B))) (some R.(and A B)))(x) }
2. Normalization
L = { (and (or (all R.(not A)) (all R.(not B))) (some R.(and A B)))(x) }
3. and-rule 적용 : (and (or (all R.(not A)) (all R.(not B))) (some R.(and A B)))(x)
L = L ∪ { (or (all R.(not A)) (all R.(not B)))(x), (some R.(and A B))(x) }
4. sume-rule 적용 : (some R.(and A B))(x)
L = L ∪ { (and A B)(y), R(x,y) }
5. and-rule 적용 : (and A B)(x)
L = L ∪ { A(y), B(y) }
6. or-rule 적용 : (or (all R.(not A)) (all R.(not B)))(x)
L1 = L ∪ { (all R.(not A))(x) }
L2 = L ∪ { (all R.(not B))(y) }
7. all-rule 적용 : (all R.(not A)), (all R.(not B))
L1 = L1 ∪ { (not A)(y) } : **모순**
L2 = L2 ∪ { (not B)(y) } : **모순**
8. Therefore, (and (not C) D) is **unsatisfiable.**

Tableau-based algorithm 에서 Labeled ABox 의 모순을 판정하는 것은 Transformation rule 적용 만큼이나 중요한 부분이다. Transformation rule 과 마찬가지로 Labeled ABox 의 모순을 판정하는 방법은 logic 의 표현 범위에 따라 조금씩 달라진다.

기본적으로, Labeled ABox 가 임의의 컨셉 P 에 대하여 P(x) 와 (not P)(x) 를 동시에 포함하고 있으면 모순이다. EOL 추론 엔진에 적용된 Tableau-based 알고리즘의 경우에는 Labeled ABox 의 모순을 판정해 낼 때, concept disjoint, 그리고 사용자 정의 포함 관계에 대한 고려가 추가로 필요하다. 때문에, Labeled ABox 가 동일한 인스턴스에 대해 disjoint concept 을 둘 다 포함하고 있는 경우와 동일한 인스턴스에 대해 상위 컨셉의 부정과 하위 컨셉을 모두 포함하고 있는 경우에도 모순이 된다.

2.2.3 추론 과정 예

Tableau-based 알고리즘을 이용해 지식 간의 포함 관계를 판정하는 과정은 다음과 같다. 예를 들어 설명하기 위해 임의의 컨셉 C, D 를 가정하였으며, R 은 롤이고, A 와 B 는 컨셉, x, y, z 는 임의의 인스턴스를, L, L1, L2, L3, L4 는 Labeled ABox 를 가리킨다. 〈표 3〉 에서는 Labeled ABox 의 확장과 그 과정에서 모순이 일어나는 과정을 보이고 있다. Transformation rule 에 따라 기존의 Labeled ABox 에 새로운 원소가 추가되며, or-rule 이 적용되는 경우에는 새로운 Labeled ABox 가 생성된다. 〈그림 2〉는 Labeled ABox 가 or-rule 에 의해 분기한 모습을 나타낸 것이다.

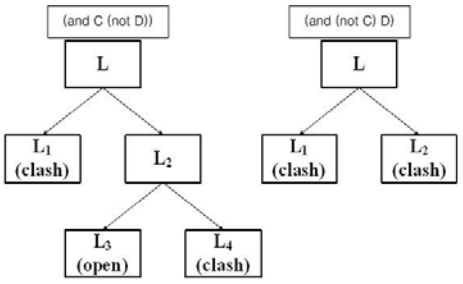

그림 2 Labeled ABox 분기 예

컨셉 C, D 간의 포함 관계를 판정하기 위해 컨셉 (and C (not D)) 와 컨셉 (and (not C) D) 의 satisfiability 를 Tableau-based 알고리즘을 이용해 판정해 보았다. 〈그림 2〉에서 알 수 있듯이, 컨셉 (and C (not D) 의 Labeled ABox 중 L3 는 모순이 발생하지 않았고, 컨셉 (and (not C) D) 의 Labeled ABox 는 모두 모순이 발생했다. 이를 이용한 포함 관계의 판정 결과는 〈표 4〉와 같다.

표 4 Subsumption Relation 판정

(and C (not D))	(and (not C) D)	Result
Satisfiable	Satisfiable	Neither C is subsumed D nor C is subsumed D.
Satisfiable	**Unsatisfiable**	**D is subsumed C.**
Unsatisfiable	Satisfiable	C is subsumed D.
Unsatisfiable	Unsatisfiable	C equals D.

3. 구현

W3C 에서 제공하는 예제 온톨로지인 Wine 온톨로지를 EOL 로 표현한 후, 이를 EOL 추론 엔진에 적용해 보았다. OWL-DL 과 달리 EOL 은 data type 을 정의하지 않는다. OWL-DL 과 EOL 로 표현된 지식을 간단히 비교해 보면, 〈표 5〉는 wine.owl 의 일부를 나타낸 것으로, Wine 클래스의 정의를 나타내고 있고, 〈표 6〉은 wine.eol 의 일부를 나타낸 것으로, Wine 컨셉의 정의를 나타내고 있다.

표 5 Definition of Wine (wine.owl)

```
〈owl:Class rdf:ID="Wine"〉
  〈rdfs:subClassOf rdf:resource="&food;PotableLiquid" /〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasMaker" /〉
      〈owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:cardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasMaker" /〉
      〈owl:allValuesFrom rdf:resource="#Winery" /〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#madeFromGrape" /〉
      〈owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:minCardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasSugar" /〉
      〈owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:cardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasFlavor" /〉
      〈owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:cardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasBody" /〉
      〈owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:cardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#hasColor" /〉
      〈owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"〉1
      〈/owl:cardinality〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:subClassOf〉〈owl:Restriction〉
      〈owl:onProperty rdf:resource="#locatedIn"/〉
      〈owl:someValuesFrom rdf:resource="&vin;Region"/〉
  〈/owl:Restriction〉〈/rdfs:subClassOf〉
  〈rdfs:label xml:lang="en"〉wine〈/rdfs:label〉
  〈rdfs:label xml:lang="fr"〉vin〈/rdfs:label〉
〈/owl:Class〉
```

표 6 Definition of Wine (wine.eol)

```
dconcept Wine
    ( and ( all hasMaker,Winery )
          ( all locatedIn,Region )
          ( atleast 1 hasSugar )
          ( atleast 1 hasFlavor )
          ( atleast 1 hasBody )
          ( atleast 1 hasColor ) )
dsubconcept Wine of PortableLiquid
```

EOL 로 표현된 지식은 간결하고 가독성이 높은 반면, OWL-DL 로 표현된 지식은 data type 과 같은, 값에 대한 제약을 포함하여 더 높은 표현범위을 제공한다. Description Logic 은 그 표현 범위가 넓어질 수록 추론 복잡도도 마찬가지로 올라가게 된다. EOL 의 목적은 온톨로지가 필요한 도메인의 지식을 표현하는데 있어서 OWL-DL 보다 표현력은 떨어지지만 가독성과 사용성을 높이는데 있다.

본 연구에서 구현한 EOL Reasoner 는 인터프리터 형식으로 입력창에 명령어를 하나씩 입력할 수 있으며, 또한 화일 입출력 기능을 제공한다. Java 를 이용해 개발하였으며, UI 는 Swing 을 이용해 구현하였다. 〈그림 3〉은 wine.eol 을 입력한 EOL 추론 엔진의 실행 화면이다.

좌측 상단의 상자는 TBox 내의 컨셉 정보를 출력하는데, 최상위 컨셉인 *TOP* 을 기준으로, 컨셉 간의 포함 관계를 나타내는 트리로 표현되어 있다. 좌측 중단의 상자는 TBox 내의 롤 정보를 출력하는데, 마찬가지로 최상위 롤인 *TOP* 을 기준으로, 롤 간의 포함 관계를 나타내는 트리로 표현되어 있다. 우측 상단의 상자는 ABox 내의 컨셉 인스턴스를 나타내고 있고, 우측 중단의 상자는 ABox 내의 롤 인스턴스를 나타내고 있다. 하단의 상자에는 EOL 이 제공하는 명령어를 이용해 질의를 요청할 경우 그 결과가 출력되거나, 기타 명령어를 입력하였을 때 정상적으로 수행되었는지 여부가 출력된다.

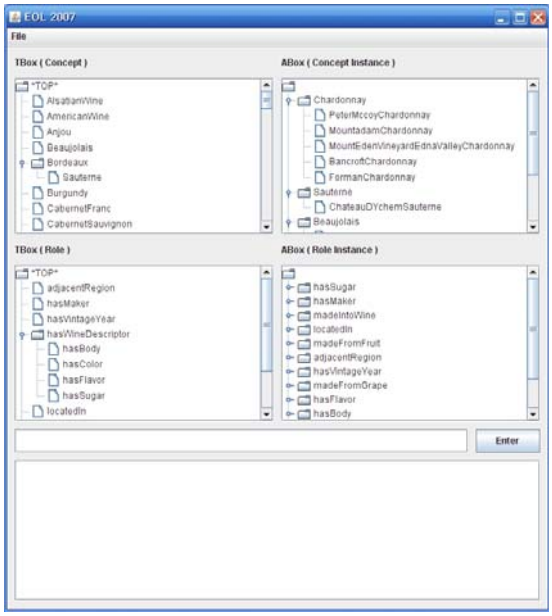

그림 3 EOL Reasoner 실행 화면

4. 결론

컴퓨팅 오브젝트의 지능적인 서비스 수행을 위해서는 지식이 필요하고, 컴퓨팅 오브젝트들이 협동을 통해 서비스를 수행할 경우, 지식들이 공유될 필요가 있다. 이러한 지식은 다시 온톨로지로 표현될 수 있는데, 본 연구에서는 온톨로지를 표현하고 처리하기 위해 Description Logic 기반의 EOL 을 이용하였다. 지식 표현 언어로 W3C 에서 제안된 OWL 은 현재 표준으로 자리 잡았지만, OWL 을 능숙하게 다루기 위해서는 숙련된 전문가가 필요하고, 그 복잡한 문법 때문에 지식을 처리하기 위해 높은 비용이 필요해진다. EOL 은 간단하고 직관적인 문법으로 비전문가도 쉽게 지식을 표현하고 활용할 수 있게 해준다. 또한 EOL 추론 엔진은 지식의 표현 수준에 따라 좀 더 좋은 성능을 발휘할 수 있는 추론 방식을 선택하기 때문에, 도메인에 따라 다양한 지식을 처리해야 하는 유비쿼터스 컴퓨팅 환경에서 사용하기에 적합하다.

참고문헌

[1] Mark Weiser, "The Computer for the 21st Century", Scientific American, Sept, 1991, pp.94~110.

[2] 이건수, 홍인표, 김민구, "EOL : SUNHI 표현 범위를 가진 인식론적 온톨로지 표현 언어", 한국정보과학회, 2006.

[3] Franz Baadar, Deborah L. McGuinnes, Daniele Nardi, Peter F. Patel-Schneider, "The Description Logic Handbook", Canibridge University Press, 2003.

[4] Micheal K.Smith, Chris Welty, Deborah L. McGuinness, "OWL Web Ontology Language Guide", W3C Recommendation, 2004.

[5] Martin Buchheit, Francesco M. Donini, Andrea Shaerf, "Decidable Reasoning in Terminological Knowledge Representation System", Journal of Artificial Intelligence Research, 1993, pp.109~138.

[6] Hector J. Levesque, Ronald J. Brachman, "Expressiveness and tractability in knowledge representation and reasoning", Computational Intelligence, 1987.

[7] F. Baadar, R. Kusters, R. Molitor, "Structural Subsumption Considered from Automata-Theoretic Point of View", International Workshop on Description Logic DL'98, 1998.

[8] Franz Baadar, Ulrike Satter, "An Overview of Tableau Algorithms for Description Logic", Kluwer Academic Publisher, 2001.

[9] 조성원, 이건수, 송세헌, 김민구, "ALKETge : 유비쿼터스 환경을 위한 지식 표현 언어", 정보과학회 추계 학술대회, 2005.

[10] Ian Horrocks, Ulrike Sattler, Stephan Tobies, "Reasoning with Individuals for the Description Logic SHIQ", proc. of the 13th Conf. on Automated Deduction(CADE-17)

[11] Ian R. Horrocks, "Optimising Tableaux Decision Procedures for Description Logics", University of Manchester, 1997.