JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 8, pp. 9-21, August 2020
https://doi.org/10.9708/jksci.2020.25.08.009

Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by R2RML Mappings

Ji-Woong Choi*

*Associate Professor, School of Computer Science and Engineering, Soongsil University, Seoul, Korea

[Abstract]

With the proliferation of RDF knowledge graphs(KGs), there arose a need of a standardized schema representation of the graph model for effective data interchangeability and interoperability. The need resulted in the development of SHACL specification to describe and validate RDF graph's structure by W3C. Relational databases(RDBs) are one of major sources for acquiring structured knowledge. The standard for automatic generation of RDF KGs from RDBs is R2RML, which is also developed by W3C. Since R2RML is designed to generate only RDF data graphs from RDBs, additional manual tasks are required to create the schemas for the graphs. In this paper we propose an approach to automatically generate SHACL schemas for RDF KGs populated by R2RML mappings. The key of our approach is that the SHACL shemas are built only from R2RML documents. We describe an implementation of our appraoch. Then, we show the validity of our approach with R2RML test cases designed by W3C.

▸ **Key words**: Knowledge graph, RDF, SHACL, R2RML, RDF validation

[요 약]

RDF 지식 그래프의 사용이 늘어나면서 표준화된 RDF 스키마 표현 형식의 부재가 데이터 상호 교환·운용성을 저해한다는 문제가 제기되어 왔다. 이를 위해 W3C는 RDF 그래프에 대한 구조 묘사 및 검증을 지원하는 SHACL 명세를 개발하였다. 관계형 데이터베이스(RDB)는 구조화된 지식 그래프를 얻는 주요 원천 중 하나이다. RDB로부터 RDF 그래프를 생성하는 방법은 통상 W3C에 의해 표준화된 R2RML 명세를 따른다. 그러나 R2RML 방식으로 생성한 RDF 그래프에 대한 스키마를 생성하기 위해서는 전문가에 의한 별도의 수작업이 요구된다. 본 논문에서는 R2RML 매핑에 의해 구축된 RDF 그래프에 대한 SHACL 스키마를 자동 생성하는 방법을 제안한다. 제안하는 방법의 특징은 R2RML 매핑 문서만으로 SHACL 스키마를 생성할 수 있다는 것이다. 본 논문은 제안하는 방법의 구현 사항들을 상세히 기술하며 구현 결과물을 W3C의 R2RML 테스트 케이스에 적용한 결과를 제시한다.

▸ **주제어**: 지식 그래프, RDF, SHACL, R2RML, RDF 검증

• First Author: Ji-Woong Choi, Corresponding Author: Ji-Woong Choi
*Ji-Woong Choi (iamjwchoi@gmail.com), School of Computer Science and Engineering, Soongsil University
• Received: 2020. 06. 08, Revised: 2020. 07. 20, Accepted: 2020. 08. 11.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

"지식 그래프"는 2012년 구글에 의해 도입된 용어이나 현재는 특정 도메인의 구조화된 지식을 그래프 형식으로 표현한 웹상에서 접근 가능한 데이터 셋을 폭넓게 일컫는다[1]. RDF(Resource Description Framework)는 지식 그래프에 대한 사실상의 표준 표현 형식으로 사용되고 있다[2]. RDF로 표현된 지식 그래프는 일반 지식 범주의 백과사전 데이터[3]에서부터 학술[4], 문화[5], 의료[6], 산업[7] 등 다양한 분야의 도메인 지식까지 광범위하게 구축되고 있다. RDF 지식 그래프의 활용 분야는 전통적으로 범용 인터넷 검색 엔진과 분야별 정보 시스템 혹은 전문가 시스템 정도였으나 최근에는 RDF 그래프를 인공 신경망의 훈련 데이터로 가공하는 기술[8-9]의 개발로 인해 이를 이용한 인공 신경망 기반의 추천 시스템[10] 및 질의응답 시스템[11]으로까지 확장되고 있다.

RDF 지식 그래프의 활용이 확산되면서, RDF 그래프에 대한 표준 스키마 포맷의 부재가 데이터 품질 관리 및 애플리케이션 통합을 어렵게 하는 주요 원인 중 하나로 꾸준히 제기되어 왔다[12-13]. 이러한 문제를 일부 수용하여 W3C는 2017년 SHACL(Shapes Constraint Language) 사양[14]을 발표하였다. SHACL의 개발로 인해 RDF 그래프의 구조를 애플리케이션 수준에서 표준화된 형식으로 교환할 수 있게 되었으며 구조 검증을 자동화할 수 있게 되었다. 따라서 RDF 그래프에 대한 SHACL 스키마의 용이한 생성을 돕는 방법의 개발이 과제로 남아 있다.

RDF 지식 그래프는 다양한 종류의 데이터로부터 정보 추출기술을 사용해 구축된다. 근래 들어 반정형·비정형 데이터로부터 자동화된 방식으로 지식 그래프를 구축하기 위한 다양한 연구가 수행되고 있으나 데이터 품질 측면에서 아직까지는 취약하다[15]. 반면, 정형데이터로부터 고품질의 RDF 그래프를 구축하는 기술은 상대적으로 성숙되어 있다. RDF 그래프를 생성하는데 사용되는 지배적인 위치의 정형데이터 타입은 관계형 데이터베이스(RDB)이다[16]. RDB로부터 RDF 그래프를 생성하는 방법은 2012년 W3C가 발표한 R2RML(RDB to RDF Mapping Language) 사양[17]에 의해 이미 표준화되어 있다. 앞서 언급한 최근까지 수행된 연구 [4-7]은 모두 이 사양을 사용하여 RDF 지식 그래프를 구축한 사례이다.

R2RML은 본래 RDB로부터 RDF 그래프만을 생성하기 위해 개발된 사양이라 생성된 그래프의 검증을 위해서는 전문가에 의한 수작업이 동반된 별도의 작업이 요구되어 왔으며 이러한 방식은 그래프의 규모가 커질수록 무결성을 보장하기 어렵다는 한계가 있다[18]. 따라서 이러한 문제를 해결하기 위하여 본 논문에서는 RDB로부터 RDF 그래프 생성을 위해 작성한 R2RML 문서를 입력으로 하여 R2RML 매핑에 의해 생성된 RDF 그래프의 구조를 설명하고 검증할 수 있는 SHACL 문서를 자동으로 생성해서 출력해 주는 방법을 제안한다. 이러한 접근법을 통해 RDF 그래프 구축 공정의 효율성과 데이터의 품질을 제고하는데 기여하고자 한다. 제안하는 방법이 그래프 생성을 위한 R2RML 매핑만 생성하면 그래프 검증에 사용할 SHACL 스키마 생성 작업을 별도로 수행할 필요가 없게 하며 자동 생성된 RDF 그래프와 SHACL 스키마를 SHACL 검증기에 입력하기만 하면 그래프 구조 검증을 마칠 수 있도록 하기 때문이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구와 배경 지식을 기술한다. 3장에서는 R2RML 문서로부터 SHACL 스키마를 자동으로 생성하는 과정과 방법을 기술한다. 4장에서는 제안하는 시스템의 테스트 결과를 제시하며, 5장에서는 결론을 맺는다.

II. Preliminaries

본 장에서는 SHACL 스키마의 용이한 생성 및 사용을 목적으로 수행된 최근 연구들을 리뷰한다. 그리고 제안하는 방법의 이해를 돕기 위해 R2RML과 SHACL 문서의 구조를 논리적 수준에서 간략히 소개한다.

1. Related works

Shape Designer[19]와 SHACL4P[20]은 RDF 그래프에 대한 스키마 저작에서부터 검증까지의 작업을 하나의 GUI 애플리케이션 상에서 가능케 하는 일종의 통합개발환경이다. SHACL4P는 대표적인 온톨로지 저작도구 중 하나인 Protégé의 플러그인 형태로 개발되었다. 즉, SHACL4P의 의의는 Protégé에서 SHACL 관련 작업을 가능케 한 것이다. Shape Designer의 특징은 SPARQL 질의문에 의해 검색된 결과 셋에 대한 스키마를 자동 생성해 준다는 것이다. 이것은 SPARQL 질의문의 구문 및 의미 분석을 통해 이를 SHACL 제약조건으로 번역해 줌으로써 이루어진다. 이때, 출력 SHACL 구문의 형식과 범위를 한정할 용도로 일종의 SHACL 구문 템플릿을 추가적인 입력으로 사용한다. SHARK[21]는 웹 기반의 RDF 그래프 검증 프레임워크로서 대규모 RDF 지식 그래프 구축 프로젝트에 적합하다. SHARK의 의의는 지속적 통합

(continuous integration) 개념을 지식 그래프 구축 프로세스에 적용했다는 데 있다. 이는 이 프레임워크가 개발 중인 RDF 그래프의 변화분에 대한 SHACL 검증 작업을 지원함은 물론 그래프의 버전 관리까지 지원함을 의미한다. shERML[22]은 RDB로부터 RDF 그래프 생성을 지원하기 위한 도구이다. 이 도구는 사용자가 GUI 환경에서 RDB 스키마를 SHACL 스키마에 매핑해 주면 RDF 그래프와 R2RML 문서를 자동 생성해준다. 본 논문에서 제안하는 시스템은 R2RML 문서가 입력이고 SHACL 문서가 출력인 반면 shERML은 SHACL 문서를 기반으로 R2RML 문서를 생성해주는 방식을 취하는 특징이 있다.

2. Overview of R2RML

```
<TriplesMap1>
 rr:logicalTable [ rr:tableName  "Student" ];
 rr:subjectMap [ rr:template "http://ex.kr/st/{ID}"; ];
 rr:predicateObjectMap [
  rr:predicate ex:role ;
  rr:objectMap [ rr:template "http://ex.kr/ro/{ROLE}"; ];
 ];
 rr:predicateObjectMap [
  rr:predicate ex:practises ;
  rr:objectMap [
   rr:parentTriplesMap <TriplesMap2>;
   rr:joinCondition [
    rr:child "Sport" ;
    rr:parent "ID" ;
   ]
  ];
 ].

<TriplesMap2>
 rr:logicalTable [ rr:tableName  "Sport" ];
 rr:subjectMap [ rr:template "http://ex.kr/sp/{ID}"; ];
 rr:predicateObjectMap [
  rr:predicate rdfs:label ;
  rr:objectMap [ rr:column "Name"; ];
 ].
```

Fig. 1. Example R2RML Mapping

R2RML 사양은 RDB 요소와 RDF 요소 간 매핑을 정의하는데 사용할 어휘와 그 어휘의 의미 그리고 준수해야 할 구문적 제약으로 구성되어있다. R2RML 사양은 R2RML 문서가 곧 RDF 문서가 되도록 설계하였다. 즉, R2RML 문서 또한 여느 RDF 문서들처럼 트리플들의 집합일 뿐이다. 이러한 이유로 R2RML 사양이 정의한 어휘들은 모두 IRI로서 R2RML 문서 작성 시 트리플의 술어 혹은 목적어로 사용된다. 목적어에 위치할 수 있는 어휘는 생략 가능하므로 실질적으로 사용되는 어휘는 모두 술어라고 봐도 무방하다. R2RML 프로세서는 R2RML 문서와 데이터베이스를 입력으로 하며 RDF 그래프들을 출력으로 한다. R2RML 프로세서는 R2RML 문서 내의 트리플들의 의미를 분석하여 매핑 규칙화한 다음 그 규칙대로 데이터베이스 데이터로부터 RDF 그래프들을 생성한다.

그림 1은 어떤 R2RML 문서의 일부분이다. 그림 1에서는 R2RML 문서의 논리 구조를 가장 직관적으로 표현해주는 Turtle 신택스[23]를 사용하였다. R2RML 문서는 논리적으로 triples map의 집합이다. triples map은 동일한 패턴의 트리플들을 생성케 하는 매핑 규칙의 단위이다. 그림 1에서는 <TriplesMap1>과 <TriplesMap2> 2개의 IRI가 triples map의 사례이다. 하나의 triples map은 1개의 logical table, 1개의 subject map, 0개 이상의 predicate object map으로 구성된다. logical table은 매핑 규칙을 적용할 DB 측 대상을 의미한다. subject map은 주어 생성 규칙, predicate object map은 술어와 목적어 생성 규칙의 쌍이다. 복수의 predicate object map은 subject map을 공유한다. 주어, 술어, 목적어 생성 규칙은 지정한 DB 측 대상의 한 행마다 적용된다. 즉, DB 측 대상의 한 행 당 최대 predicate object map의 개수만큼 트리플이 생성되며 이들은 주어를 공유한다. DB 측 행 데이터 내에 트리플 생성을 위해 지정된 컬럼값이 널이면 매핑을 적용하지 않으므로 한 행 당 predicate object map의 개수만큼 트리플이 생성되지 않을 수 있다.

그림 1에서 술어 rr:logicalTable의 목적어 트리플이 logical table의 사례이다. Student와 Sport 테이블이 각각 <TriplesMap1>과 <TriplesMap2>을 위한 DB 측 대상으로 취해졌다. 그림 1에서 술어 rr:subjectMap의 목적어 트리플이 subject map이다. 이 subject map들은 모두 템플릿을 사용하고 있다. 술어 rr:template의 목적어인 템플릿 문자열 내의 중괄호 파트가 DB 데이터로 치환될 부분이다. 중괄호 안의 문자열은 컬럼명이다. 예들 들어, <TriplesMap1>의 경우 템플릿 문자열의 '{ID}' 부분이 Student 테이블의 매 행으로부터 취해진 널이 아닌 ID 컬럼값으로 치환된다. 치환된 문자열은 주어 자격의 IRI가 된다. R2RML 사양은 템플릿으로부터 생성된 문자열에 대한 디폴트 타입을 IRI로 한다. 그림 1에서 술어 rr:predicateObjectMap의 목적어가 predicate object map의 사례들이다. 생성할 트리플의 술어와 목적어 생성 규칙은 predicate object map의 술어 rr:predicate과 rr:objectMap의 목적어 자리에 각각 위치한다. 그림 1에서는 rr:predicate의 목적어로 모두 IRI 타입의 상수가 사용되었으며 가장 빈도가 높은 사용 사례이다. 이 경우 사용된 IRI는 변환과정 없이 그대로 생성되는 트리플의 술어가 된다. 그림 1은 세 가지 목적어 생성 규칙 사례를 보여준다. 우선 ex:role의 목적어를 위한 생성 규칙은 Student 테이블의

ROLE 컬럼값이 치환되어 IRI를 생성케 한다. 다음으로 ex:practises의 목적어를 위한 생성 규칙은 referencing object map의 사례이다. 이 object map은 다른 triples map에 의해서 생성된 트리플의 주어를 목적어로 취하게 한다. 그림 1에서는 <TriplesMap2>에 의해 생성된 주어가 ex:practises의 목적어가 된다. 이 object map의 술어 rr:joinCondition 이하에 의해서 Sport 테이블의 ID 컬럼값이 Student 테이블의 Sport 컬럼값과 같은 행으로부터 <TriplesMap2>에 의해서 생성된 주어들이 ex:practises의 목적어가 된다. 마지막으로 rdfs:label의 목적어를 위한 생성 규칙은 Sport 테이블의 Name 컬럼값을 리터럴 타입의 목적어로 취하게 한다. R2RML 사양은 rr:column에 의해서 생성된 목적어의 디폴트 타입을 리터럴로 한다.

3. Overview of SHACL

```
ex:PersonShape
 a sh:NodeShape ;
 sh:nodeKind sh:IRI;
 sh:property [
  sh:path ex:ssn ;
  sh:maxCount 1 ;
  sh:datatype xsd:string ;
  sh:pattern "^₩₩d{3}-₩₩d{2}-₩₩d{4}$" ;
 ] ;
 sh:property ex:WorksForShape .

ex:WorksForShape
 a sh:PropertyShape ;
 sh:path ex:worksFor ;
 sh:class ex:Company ;
 sh:nodeKind sh:IRI .
```

Fig. 2. SHACL Example

SHACL 사양은 RDF 그래프의 임의의 노드에 대한 구조를 묘사할 수 있는 어휘들을 제공한다. SHACL 사양을 준수하여 작성한 SHACL 문서는 R2RML 문서와 같이 RDF 문서가 된다. 따라서 SHACL 문서 내의 트리플들은 RDF 그래프를 구성하게 되며 이를 shapes 그래프라고 한다. shapes 그래프를 가지고 임의의 RDF 그래프의 형상을 검증할 수 있다. 검증 대상 그래프는 data 그래프라고 한다. SHACL processor는 SHACL 검증기(validator)를 의미하며 shapes 그래프를 가지고 data 그래프의 형상을 검증한다. 이때의 shapes 그래프는 data 그래프 형상에 대한 제약조건으로 해석된다.

그림 2는 Turtle 신택스로 표현된 shapes 그래프의 예이다. shapes 그래프를 구성하는 의미 단위를 shape이라고 한다. shape은 node shape과 property shape 중 하나이다. node shape은 RDF 그래프의 노드의 특성을 제약하며 property shape은 노드가 포함된 트리플의 구조를 제약한다. 그림 2에서 ex:PersonShape이 node shape, ex:WorksForShape이 property shape이다. property shape은 node shape의 술어 sh:property의 목적어 자리에서 직접 정의될 수도 있고 참조될 수도 있다. ex:PersonShape에서 보이는 2개의 sh:property의 목적어 형식이 이에 대한 각각의 예다. 그림 2의 의미는 다음과 같다. ex:PersonShape에 부합하는 RDF 노드는 우선 IRI여야 한다. 그리고 최대 1개의 ex:ssn, 0개 이상의 ex:worksFor를 술어로 하는 트리플의 주어여야 한다. 참고로, SHACL은 sh:maxCount 혹은 sh:minCount로 제약되지 않은 property shape의 cardinality는 0 이상으로 한다. ex:ssn의 목적어는 문자열(xsd:string) 리터럴이어야 하며 숫자 3자리, 대시, 숫자 2자리, 대시, 숫자 4자리의 패턴을 만족시켜야 한다. ex:worksFor의 목적어는 ex:Company의 인스턴스 IRI여야 한다.

III. The Proposed Method

본 장에서는 R2RML 문서로부터 SHACL 문서를 생성하기 위한 방법을 처리 순서에 따라 3단계로 나누어 기술한다. 첫 단계에서는 R2RML 문서로부터 R2RML 모델을 복원한다. 다음 단계에서는 R2RML 모델로부터 SHACL 모델을 생성한다. 마지막 단계에서는 SHACL 모델로부터 SHACL 문서를 출력한다.

1. Constructing an R2RML Model

이 단계에서는 R2RML 문서를 입력으로 취하여 R2RML 모델을 시스템 내부에 구축한다. R2RML 모델은 R2RML 문서가 내포하는 논리 모델을 시스템 내부에서 복원시킨 것이다. 이를 위하여 입력 문서에 대한 두 가지 수준의 복원 작업이 차례로 수행된다. 우선 입력 문서를 RDF 수준으로 복원한다. 그 이유는 R2RML 문서가 RDF 문서라는 사실에 의거한다. 즉, 모든 R2RML 문서는 RDF 문서로 다룰 수 있기 때문이다. 이 수준의 복원은 R2RML 문서로부터 시스템 내부에 RDF 모델의 생성이 이루어졌음을 의미하며 R2RML 문서의 요소들을 트리플 단위로 접근할 수 있게 되었음을 의미한다. 이 수준의 작업은 오픈소스 프로젝트인 Apache Jena의 RDF API를 기반으로 수행하였다. 이어서 R2RML 수준의 복원 작업이 수행된다. 즉, RDF 모델로부

터 R2RML 모델을 시스템 내부에 구축함을 의미한다. RDF 모델 내의 각각의 트리플들의 의미 관계는 R2RML 논리 모델을 따른다. 따라서 R2RML 논리 모델의 의미 구조에 의거한 연관된 트리플들의 검색 그리고 검색된 트리플 내에서 R2RML 모델의 구축에 필요한 요소들의 추출이 가능해야 R2RML 모델을 구축할 수 있다. 이를 위하여 R2RML 파서 모듈을 자체 제작하였으며 이것을 사용하여 R2RML 문서로부터 R2RML 모델이 최종적으로 복원된다.

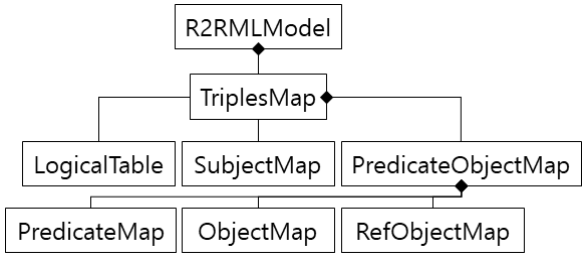

Fig. 3. Major Classes in R2RML Model

그림 3은 R2RML 모델의 핵심 클래스들의 관계를 보여준다. 그림 3은 본 논문의 R2RML 모델이 R2RML 사양의 논리 모델을 왜곡 없이 반영함을 보여준다. R2RML 문서 내의 요소와 R2RML 모델 객체는 1:1 대응되며 객체 사이의 대응 관계 제약조건도 R2RML 사양을 준수한다. 1개의 R2RML 문서로부터 1개의 R2RMLModel 객체가 생성된다. R2RMLModel 객체는 복수의 TriplesMap 객체를 소유할 수 있다. TriplesMap 객체는 반드시 1개씩의 LogicalTable, SubjectMap 객체를 소유해야 하며 0개 이상의 PredicateObjectMap 객체 소유가 가능하다. PredicateObjectMap 객체는 1개 이상의 PredicateMap 객체, 0개 이상의 ObjectMap 객체, 0개 이상의 RefObjectMap 객체를 소유할 수 있다. 다만, ObjectMap과 RefObjectMap 두 타입의 객체 모두 0개 일 순 없다.

그림 4는 R2RML 모델 클래스들의 세부 속성들을 보여준다. 각각의 속성 옆에는 주석에 의해 해당 속성이 R2RML 문서의 어떤 속성에 대응하는지도 표기하였다. 그림 4의 TermMap은 SubjectMap, PredicateMap, ObjectMap의 수퍼 클래스이다. 이들 4개의 클래스 관계는 R2RML 사양 정의를 따른 것이다. SubjectMap은 TermMap의 속성을 상속함은 물론 추가적으로 classes 속성을 갖는다. classes 속성은 IRI의 집합이다. 이것은 rr:class의 값이 IRI만 가능하며 복수일 수 있기 때문이다. rr:class의 값들은 생성될 트리플 주어의 타입(rdf:type)으로 사용된다.

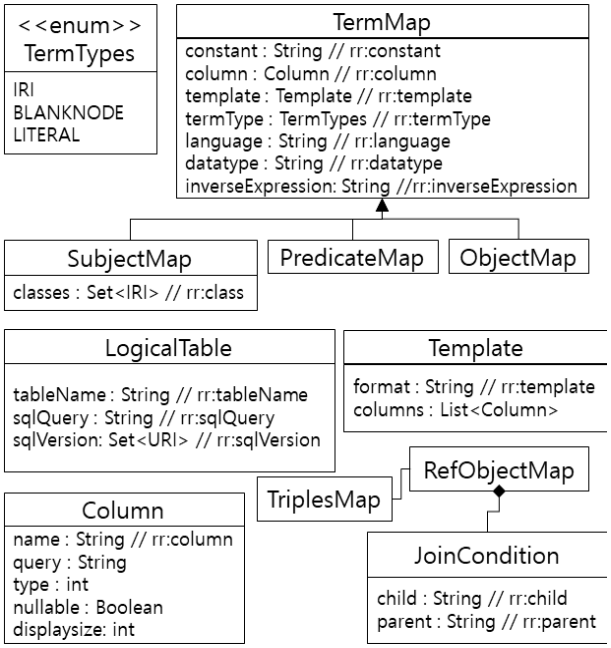

Fig. 4. Properties of Classes in R2RML Model

그림 4에서 R2RML 사양에는 정의되어 있지 않으나 추가적으로 생성한 클래스는 TermTypes, Template, Column이다. TermType의 열거형 상수 IRI, BLANKNODE, LITERAL은 R2RML 문서의 rr:termType의 값으로 가능한 rr:IRI, rr:BlankNode, rr:Literal에 각각 대응한다. Template 클래스의 format 속성값은 rr:template의 값인 정규 표현식 문자열이며 columns 속성은 SHACL 모델 생성의 편의를 위해 정규 표현식 내에 포함된 컬럼명들 만을 추출하여 목록으로 따로 보관한다. Column 클래스의 name 속성값은 rr:column의 값인 컬럼명 문자열이다. Column 클래스의 나머지 속성들 또한 SHACL 모델 생성 시 해당 컬럼에 대하여 요구되는 추가 정보들을 미리 마련해 두고 구조화시킨 것들이다. query 속성은 해당 컬럼의 값을 추출할 수 있는 SQL 질의문이며 type 속성은 컬럼의 SQL 데이터 타입, nullable 속성은 컬럼의 널 허용 여부, displaysize 속성은 문자열 타입에 한한 것으로 최대 가능 문자 개수이다. 이러한 컬럼에 대한 메타데이터들은 직접 DB에 접근하여 추출하였다.

2. Constructing a SHACL Model

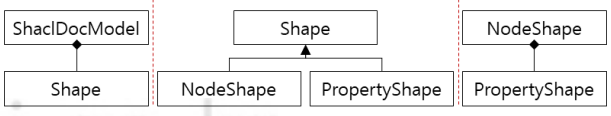

Fig. 5. Class Relationships in SHACL Model

두 번째 단계에서는 이전 단계에서 구축한 R2RML 모델을 입력으로 하여 SHACL 모델을 구축한다. SHACL 모델은 그림 5에서 ShaclDocModel 타입의 객체에 대응한다. ShaclDocModel 객체는 Shape 객체들의 집합이다. Shape은 구체적으로 NodeShape 혹은 PropertyShape이다. Shape 클래스에 대한 상속 관계는 SHACL 사양의 정의를 따라 설계하였다. NodeShape은 RDF 트리플의 노드 즉, 주어 혹은 목적어 그 자체의 성질에 대한 제약 조건만을 명세한다. PropertyShape은 특정 RDF 노드를 중심으로 그 노드가 포함된 트리플 구조에 대한 제약 조건을 명세한다. 그림 5의 가장 우측에 표현된 NodeShape과 PropertyShape의 관계는 하나의 NodeShape이 복수개의 PropertyShape의 참조를 갖음을 의미한다. 참조된 PropertyShape들은 NodeShape 제약 조건을 만족하는 특정 노드가 함께 만족해야 하는 트리플 구조에 대한 제약 조건들이 된다.

Table 1. Input for Building SHACL Model

Output	Input	
SHACL Model	R2RML Model	SHACL Model
NodeShape (Type1)	SubjectMap	
NodeShape (Type2)		set of Type1 NodeShapes
PropertyShape (Type1)	PredicateMap, ObjectMap	
PropertyShape (Type2)	PredicateMap, RefObjectMap	

표 1은 R2RML 모델과 SHACL 모델의 요소 간 대응 관계를 요약한다. SHACL 모델 요소들은 대응하는 R2RML 요소를 입력으로 하여 R2RML 요소 내의 속성을 분석하여 SHACL 요소의 세부 속성을 결정한다. 그림 6은 두 모델 요소 간 짝지어지는 순서를 설명한다. NodeShape과 SubjectMap의 대응수는 1:1이다. SubjectMap은 TriplesMap 내에서 1개만 존재할 수 있기 때문에, TriplesMap 마다 1개의 NodeShape이 생성된다.

PropertyShape은 1개의 술어-목적어 쌍으로부터 생성된다. 여기서 술어는 1개의 PredicateMap을 의미하고 목적어는 1개의 ObjectMap 혹은 1개의 RefObjectMap을 의미한다. 대응된 목적어 타입을 기준으로 기술의 편의를 위하여 Type1, Type2 PropertyShape으로 구분하였다. 1개의 R2RML PredicateObjectMap으로부터 다수의 술어-목적어 쌍이 생성될 수 있다. PredicateObjectMap이

```
function buildSHACLModel(r2rml)
input: R2RML Model r2rml
output: SHACL Model shacl
01  shacl ← create an empty SHACL Model.
02  TMs ← get the set of TriplesMaps in r2rml.
03  for each TM ∈ TMs {
04    SM ← get the SubjectMap of TM.
05    NS ← create a type 1 NodeShape with SM.
06    POMs ← get the set of PredicateObjectMaps.
07    for each POM ∈ POMs {
08      POPairs ← create the predicate-object pairs
                  from POM.
09      for each POPair ∈ POPairs {
10        PM ← get the PredicateMap of POPair.
11        OM ← get the ObjectMap of POPair.
12        ROM ← get the RefObjectMap of POPair.
13        if OM != null {
14          PS ← create a type 1 PropertyShape with
                 PM and OM.
15        }
16        if ROM != null {
17          PS ← create a type 2 PropertyShape with
                 PM and ROM.
18        }
19        add PS to shacl and NS respectively.
20      }
21    }
22    add NS to shacl.
23  }
24  C ← create a collection of sets, elements of each
    of which are all type 1 NodeShapes which can be
    used to explain an RDF subject node together.
25  for each s ∈ C {
26    NS2 ← create a type 2 NodeShape with s.
27    add NS2 to shacl.
28  }
29  return shacl
```

Fig. 6. Algorith for building SHACL model

n개의 PredicateMap, m개의 ObjectMap, p개의 RefObjectMap으로 구성되어 있다면, n개의 술어와 m+p개의 목적어로 구성된 것으로 해석하여 n(m+p)개의 술어-목적어 쌍이 도출된다. 생성된 PropertyShape들은 자신을 생성케 한 PredicateObjectMap이 속한 TriplesMap과 동일한 TriplesMap을 기반으로 생성된 Type1 NodeShape에 포함된다. 그림 6의 23 라인까지 실행되면 R2RML 모델의 모든 TriplesMap과 그 하위 요소에 대한 1회씩의 방문을 통해 Type2 NodeShape을 제외한 모든 Shape의 생성과 등록이 완료된다.

그림 6의 24~28 라인은 Type2 NodeShape의 생성 절차를 기술한다. Type2 NodeShape은 R2RML 변환이 서로 다른 TriplesMap에서 동일한 주어 노드를 생성할 수도 있다는 특성에 대비한 것이다. 예를 들어, 동일한 주어 노드를 생성할 수 있는 TriplesMap의 집합 {A, B, C}가 있다면, 트리플 생성 결과 동일한 주어 노드를 생성한

TriplesMap의 조합은 {A}, {B}, {C}, {A, B}, {B, C}, {A, B, C}가 가능하다. 이 상황에서 {A, B}, {B, C}, {A, B, C} 조합에 의해 생성된 트리플들의 형상 검증은 A 혹은 B 혹은 C에 대응하여 생성한 개별 Type1 NodeShape 만으로는 일부 트리플들만 검증하게 되므로, 둘 혹은 세 개의 Type1 NodeShape의 제약조건 모두를 논리적으로 결합한 NodeShape이 필요하며 이것이 Type2 NodeShape이다. 따라서, 동일한 주어 노드를 생성할 가능성이 있는 TriplesMap의 집합의 원소가 n개라면 이 집합의 부분집합 중에서 원소의 개수가 2개 이상인 부분집합 각각에 대응하여 Type2 NodeShape이 생성된다. Type2 NodeShape은 대응하는 부분집합의 원소들 각각으로부터 생성한 Type1 NodeShape들의 집합으로부터 생성된다.

Table 2. Constraints of Type1 NodeShape

NodeShape	SubjectMap
sh:nodeKind	rr:termType
sh:class	rr:class
sh:hasValue	rr:constant
sh:pattern	rr:template

표 2는 Type1 NodeShape을 구성하는 하위 제약조건들이 R2RML SubjectMap의 어떤 하위 속성으로부터 유도되는지를 보여준다. sh:nodeKind는 타겟 노드의 타입을 제한한다. 타겟 노드는 해당 제약조건에 의해 평가되는 RDF 노드를 일컫는다. sh:nodeKind는 rr:termType이 블랭크 노드로 명시되지 않는 한 IRI로 결정된다. 이것은 R2RML 명세의 R2RML 프로세서를 위한 노드 타입 결정 알고리즘을 동일하게 구현한 것이다. sh:class는 타겟 노드가 특정 RDF 클래스의 인스턴스이어야 함을 제한한다. sh:class는 rr:class가 명세 되어 있을 때만 존재할 수 있으며 sh:class의 값은 rr:class의 값과 늘 동일하다. 그리고 그 개별 값의 타입은 IRI다. sh:hasValue는 타겟 노드가 특정된 IRI여야 함을 제한하며 rr:constant가 명세 되어 있는 경우만 사용된다. 그리고 rr:constant의 값이 sh:hasValue의 값으로 그대로 쓰인다. sh:pattern은 타겟 노드 문자열이 특정 정규표현식을 만족하여야 함을 의미한다. sh:pattern은 rr:template이 사용되었을 때 정의할 수 있으며 sh:pattern의 값은 정규표현식으로서 rr:template의 문자열 중 컬럼명 부분만을 임의의 문자가 여러 번 올 수 있음을 의미하는 문자열 '.*'으로 치환한 것이다.

Table 3. Constraints of Type1 PropertyShape

PropertyShape	ObjectMap
sh:nodeKind	rr:termType
sh:in	rr:constant, rr:object
sh:languageIn	rr:language
sh:datatype	rr:datatype
sh:maxLength	rr:column
sh:pattern	rr:template
sh:minCount	rr:constant, rr:object, rr:column, rr:template
sh:maxCount	rr:constant, rr:object

표 3은 PropertyShape의 하위 제약조건들 목록과 이 조건들의 값 결정에 영향을 끼치는 ObjectMap 하위 속성들의 대응 관계를 보여준다. 표 3의 제약조건들은 타겟 노드에 목적어 자격으로 연결된 노드에 대한 제약을 표현한다. sh:nodeKind는 목적어 노드의 타입을 제약한다. sh:nodeKind는 rr:termType의 값이 명시되었다면 rr:termType의 값을 그대로 취한다. rr:termType이 사용되지 않았다면 R2RML 명세의 목적어 노드 타입 결정 알고리즘을 따라서 sh:nodeKind 값을 결정한다. 이 알고리즘을 요약하면 R2RML 목적어 노드의 명세가 rr:constant 혹은 rr:object를 통해 이루어졌고 이들 속성의 값 타입이 IRI이면 IRI로 결정되고 그 외의 경우는 리터럴로 결정된다. sh:in은 목적어 노드의 후보 값들을 구체적으로 나열함으로써 값의 범위를 제한한다. 그 후보 값은 rr:constant 혹은 rr:object의 속성값을 그대로 취한다. sh:languageIn은 목적어 리터럴의 언어 태그를 제한하며 rr:language의 값을 그대로 취한다. sh:datatype은 목적어 리터럴의 데이터 타입을 제한한다. sh:datatype의 값은 rr:datatype이 사용되었다면 그 값을 그대로 취한다. 그러나 rr:datatype이 사용되지 않았더라도 rr:column에 의해 목적어 생성을 정의한 경우 R2RML의 natural mapping of SQL values 규칙을 적용하여 sh:datatype의 값을 결정했다. 이 규칙은 SQL 데이터 타입과 XML 데이터 타입의 대응 관계를 정의한 것이다. 따라서 rr:column의 값으로 사용된 컬럼의 SQL 데이터 타입을 구한 후 대응하는 XML 데이터 타입을 sh:datatype의 값으로 결정하며 이 과정에서 데이터베이스 스키마 정보가 사용된다. sh:maxLength는 목적어 노드가 문자열 리터럴일 경우 문자열 길이의 상한을 제약한다. 이 제약조건은 rr:column이 사용에 의한 문자열 리터럴 목적어가 생성될 경우에 사용된다. sh:maxLength의 값은 rr:column의 값으로 사용된 컬럼의 데이터베이스 스키마 정보를 참조하여 얻은 해당 컬럼의 최대 문자 개수가 된다.

sh:pattern은 목적어 노드가 만족시켜야 하는 정규 표현식이다. sh:pattern의 값은 rr:template의 값 중 컬럼명 파트들을 '.*' 혹은 '.{0, n}'으로 치환한 값이다. 그 중에서 형식 '.{0, n}'은 해당 컬럼의 SQL 타입이 문자열일 경우에 한하여 사용되며 n은 sh:maxLength의 값 결정 방식과 동일한 방식으로 얻은 값이 사용된다. sh:minCount와 sh:maxCount는 목적어 노드의 cardinality 제약조건이다. 본 논문에서 생성하게 되는 cardinality는 세 가지이며 1, 0 이상, 1 이상이다. SHACL 명세는 sh:minCount가 생략되었을 경우 0 이상으로 간주하므로 이 경우 별도의 출력이 없도록 했다. rr:column 혹은 rr:template의 값으로 사용된 컬럼이 null을 허용하면 0 이상이 되며 그렇지 않다면 1 이상이 된다. 특히, rr:template은 복수의 컬럼일 수 있으므로 모든 컬럼이 null을 허용하지 않아야 1 이상이 된다. rr:constant와 rr:object가 사용되었다면 cardinality는 1이다.

Type2 Shape은 Type1 Shape의 논리적 결합으로 구성되며 구체적 신택스는 다음절에서 기술한다. 다만, 구성 요소로서의 Type1 Shape들은 모두 sh:pattern 값이 같은 특징이 있다. 이것이 같은 주어 노드를 공유할 가능성이 있음을 의미하기 때문이다.

3. Writing a SHACL Document

그림 7~11은 모든 유형의 출력 Shape 구문을 보여준다. 출력 문서는 이 5가지 구문 유형들의 나열일 뿐이다.

```
01  IRI                                                    1
02    a sh:NodeShape ;                                     1
03    sh:nodeKind sh:IRI | sh:BlankNode ;                  1
04    sh:class IRI ;                                       *
05    sh:hasValue IRI ;
06    sh:pattern "regex" ;
07    sh:property IRI  .                                   *
```

Fig. 7. Type1 NodeShape Syntax I

```
01  IRI                                                    1
02    a sh:PropertyShape ;                                 1
03    sh:path IRI ;                                        1
04    sh:nodeKind sh:Literal | sh:IRI | sh:BlankNode ;     1
05    sh:in ( IRI | literal ) ;
06    sh:languageIn ( literal ) ;
07    sh:datatype IRI ;
08    sh:pattern "regex" ;
09    sh:minCount 1 ;
10    sh:maxCount 1 ;
11    sh:maxLength nonnegative_integer .
```

Fig. 8. Type1 PropertyShape Syntax I

```
01  IRI                                                    1
02    a sh:PropertyShape ;                                 1
03    sh:path IRI ;                                        1
04    sh:node IRI .                                        1
```

Fig. 9. Type2 PropertyShape Syntax

```
01  IRI                                                    1
02    a sh:PropertyShape ;                                 1
03    sh:path IRI ;                                        1
04    sh:qualifiedValueShape [                             1
05      sh:nodeKind ~ ; sh:in ~ ; sh:languageIn ~ ;
06      sh:datatype ~ ; sh:pattern ~ ; sh:maxLength ~ ;
07    ]                                                    1
08    sh:qualifiedMinCount 1 ;
09    sh:qualifiedMaxCount 1 ;
10    sh:qualifiedShapeDisjoint true .                     1
```

Fig. 10. Type1 PropertyShape Syntax II

```
01  IRI                                                    1
02    a sh:NodeShape ;                                     1
03    sh:and (                                             1
04      [ sh:qualifiedValueShape IRI ].                    2+
05    ) .                                                  1
```

Fig. 11. Type2 NodeShape Syntax

그림 7은 Type1 NodeShape, 그림 11은 Type2 NodeShape 구문이다. 그림 8과 그림 10은 Type1 PropertyShape 구문이며 그림 9는 Type2 PropertyShape 구문이다. 그림 7~11의 3열은 각각의 행 구문이 몇 번 출현할 수 있는가를 표현한다. '1'은 1회로써 필수 요소임을 의미한다. 공백은 0 또는 1회, '*'은 0회 이상을 의미하며 선택 요소이다. 이 선택 요소들의 사용 횟수는 모두 해당 요소를 생성케 하는 R2RML 요소의 사용 여부 혹은 사용 횟수와 같아진다. '2+'는 2회 이상을 의미한다.

그림 7~11의 1행에 있는 IRI는 모두 Shape 식별자이다. 그림 7~11의 2행은 식별자의 타입 선언이다. 타입은 NodeShape 혹은 PropertyShape이다. 선언된 식별자들은 다음 세 가지 경우에 다른 Shape에서 참조된다. 첫 번째로 모든 PropertyShape 식별자는 Type1 NodeShape에서 참조된다. 참조 위치는 그림 7의 7행이다. 참조하는 NodeShape이 주어 노드를 제약하고 참조된 PropertyShape들은 그 주어의 술어와 목적어 노드를 제약하게 된다. 따라서 그림 7의 7행은 해당 주어 노드에 연결될 수 있는 모든 술어, 목적어 형상들의 쌍의 수만큼 반복되며 반복횟수는 해당 Type1 NodeShape을 생성케 한 TriplesMap 내의 술어-목적어 쌍의 개수와 같다. 두 번째로 Type2 PropertyShape은 Type1 NodeShape을 참조한다. 참조 위치는 그림 9의 4행이다. 참조되는 NodeShape은 참조하는 Type2 PropertyShape을 생성케 RefObjectMap 내의 rr:parentTriplesMap에 의해 지

정된 TriplesMap에 대응하여 생성된 Type1 NodeShape이다. 세 번째로 동일한 주어 노드를 공유할 수 있는 Type1 NodeShape들은 Type2 NodeShape에서 참조될 수 있다. 참조 위치는 그림 11의 4행이다. 그림 11의 4행의 반복횟수는 2회 이상인데 이것은 Type2 NodeShape의 생성 시 입력되는 Type1 NodeShape 집합의 최소 원소 수가 2이기 때문이며 1개의 Type1 NodeShape 식별자가 한 행씩 차지하는 형식으로 반복된다.

그림 11은 sh:qualifiedValueShape 제약 조건 하에 있는 NodeShape들이 sh:and 논리로 묶여있는 구조이다. sh:qualifiedValueShape은 각각의 NodeShape을 다른 NodeShape의 간섭없이 독립적으로 타겟 노드 검증에 사용하게 하는 효과가 있다. 이것은 sh:and 내부의 서로 다른 NodeShape 간 모순된 제약조건이 존재할 수 있기 때문이다. 따라서 타겟 노드가 sh:and 안에 나열된 NodeShape들을 모두 그러나 독립적으로 각각 만족시키면 검증이 성공하는 구조이다.

그림 10은 Typ1 PropertyShape 구문으로서 특정한 상황에서 그림 8의 Type1 PropertyShape 구문 대신 사용된다. 특정한 상황이란 주어와 술어는 같으나 목적어가 서로 다른 트리플들이 존재하는 상황이다. 주어는 NodeShape으로 검증된다. 술어와 목적어는 해당 NodeShape의 sh:property에서 참조한 PropertyShape으로 검증된다. PropertyShape는 sh:path 값과 술어가 같다면 그 술어의 목적어를 검증한다. 그러나 sh:path가 같은 PropertyShape이 여러 개 존재하기 때문에 검증은 반드시 실패하게 된다. 즉, 한 PropertyShape은 만족시키더라도 다른 ProperyShape은 만족시킬 수 없기 때문이다. 이러한 상황을 해결하기 위하여 하나의 술어, 목적어 쌍이 동일한 sh:path의 PropertyShape들 중에서 최소한 하나의 PropertyShape 검증을 통과하기만 하면 그 트리플이 검증을 통과한 것으로 판단되도록 논리를 조정하였다. 이 경우 그림 8의 제약 조건들 중에서 그림 10의 5, 6행에 나열된 제약 조건들은 그림 10에서 처럼 sh:qualifiedValueShape 블록 내부로 옮겨져서 출력되며 sh:minCount와 sh:maxCount은 그림 10의 8, 9행 처럼 qualified가 첨가된 제약조건으로 대치되어 출력된다.

IV. Experiments

제안하는 시스템의 유효성을 검증하기 위하여 W3C의 “R2RML and Direct Mapping Test Cases”[24]를 활용하였다. R2RML과 Direct Mapping은 모두 RDB 데이터로부터 RDF 데이터를 생성하기 위한 용도로 W3C가 개발한 표준 사양의 이름이다. 테스트 케이스 문서는 87개의 테스트 케이스를 D000부터 D025로 명명한 26가지 카테고리로 분류하여 정의하고 있다. 각각의 카테고리는 RDB 구조에 따른 분류이다. 24개의 테스트 케이스는 Direct Mapping, 63개의 테스트 케이스는 R2RML을 위한 것이다. 본 논문에서는 R2RML을 위한 63개의 테스트 케이스만을 활용 대상으로 한정하였다. 각각의 R2RML 테스트 케이스는 RDB의 생성을 위한 SQL 스크립트, R2RML 문서 그리고 RDF 문서로 구성되어 있다. 본래 이 테스트 케이스의 검증 대상은 R2RML 사양을 구현한 시스템(프로세서)이기 때문에, 테스트는 검증 대상 프로세서가 각각의 테스트 케이스의 SQL 스크립트에 의해 생성된 RDB와 R2RML 문서를 입력으로 취하여 RDF 문서를 생성케 하며 이것이 테스트 케이스에서 제시한 RDF 문서와 같다면 테스트 통과로 판정한다. 그러나 본 논문에서의 검증 대상은 SHACL 생성기이기에 테스트 방식과 테스트 기준을 달리하였다. 테스트 방식은 SHACL 생성기가 각각의 테스트 케이스에서 주어진 RDB와 R2RML 문서를 입력으로 취하여 SHACL 문서를 출력하게 한 후 출력 SHACL 문서를 가지고 테스트 케이스에서 제시한 RDF 문서를 검증케 하였다. 테스트 기준은 SHACL 문서가 RDF 문서 내의 모든 트리플의 구조를 빠짐없이 설명할 수 있을 때 통과로 판정하였다. 본 테스트에서 사용한 SHACL 검증기는 dotNetRDF이며 이 검증기는 2019년 W3C에 의해서 수행된 SHACL 검증기 테스트[25]에서 모든 항목을 통과한 2개의 검증기 중 하나이다. 또한 MySQL 소스 기반의 오픈소스 DBMS인 MariaDB를 테스트 DBMS로 사용하였다.

Table 4. Summary of Test Results

Categories	Used & Passed Test Cases	Categories	Used & Passed Test Cases
D000 (1)		D010 (3)	a, b, c
D001 (2)	a, b	D011 (2)	a, b
D002 (10)	a, b, d, i, j	D012 (5)	a, b, e
D003 (3)	b, c	D013 (1)	a
D004 (2)	a	D014 (4)	a, b, c, d
D005 (2)	a, b	D015 (2)	a
D006 (1)	a	D016 (5)	a, b, c, d, e
D007 (8)	a, b, c, d, e, f, g	D018 (1)	a
D008 (3)	a, b, c	D019 (2)	a
D009 (4)	a, b, c, d	D020 (2)	a

표 4는 테스트 결과를 요약한다. 표 4의 1, 3열은 카테고리 이름이며 괄호 안의 숫자는 해당 카테고리에 정의된 테스트 케이스 개수이다. 표 4의 2, 4열은 각각의 카테고리에서 테스트에 사용되었으며 테스트를 통과한 테스트 케이스 이

름이다. 결과를 요약하면, R2RML을 위한 63개의 테스트 케이스 중 49개의 테스트 케이스를 가지고 테스트를 수행했으며 모든 테스트 케이스의 RDF 문서에 포함된 트리플들의 구조를 본 연구가 제안한 시스템을 통해 생성한 SHACL 제약조건을 통해 모두 설명 가능하였다. 63개 테스트 케이스 중 본 테스트에서 제외된 테스트 케이스는 14개이다. 이들 중 D000에 정의된 테스트 케이스 a는 RDF 문서 내에 트리플의 개수가 0개이다. 즉, 빈 RDF 문서의 생성을 테스트하기 위한 테스트 케이스이다. 따라서, 검증할 트리플이 존재하지 않으므로 테스트에서 제외하였다. 나머지 13개의 테스트 케이스는 RDF 문서의 생성이 불가능함을 테스트하기 위한 테스트 케이스들이다. 따라서, 검증할 RDF 문서가 존재하지 않기 때문에 테스트에서 제외하였다.

그림 12~14 그리고 그림 15~17로 구성된 두 개의 테스트 케이스 사례를 통해 본 연구의 유효성을 설명하고자 한다. 그림 12~14는 그림 10의 신택스 즉, Type1 PropertyShape 신택스 II가 RDF 구조 검증에 있어서 효과를 발휘하는 사례이다. 그림 15~17은 그림 11의 신택스 즉, Type2 NodeShape이 RDF 구조 검증에 있어서 효과를 발휘하는 사례이다.

그림 12는 D007 카테고리에 정의된 테스트 케이스 d의 R2RML 매핑 정의이고 그림 13은 이 정의에 의해 생성된 RDF 트리플들이다. 그림 14는 본 연구의 시스템이 그림 12의 R2RML 매핑 정의를 입력으로 하여 생성한 SHACL 문서이다. 그림 12는 1개의 triples map ①이 정의되어 있으며 그 안에 4개의 predicate object map ②~⑤가 정의되어 있다. 이를 반영하여 그림 14의 SHACL 문서는 1개의 node shape ①과 4개의 property shape ②~⑤로 구성되어 있다. 그림 12의 subject map에 정의된 주어 노드 생성 규칙은 그림 14의 node shape ① 내의 sh:nodeKind와 sh:pattern 제약조건으로 번역되어 있다.

```
<TriplesMap1> // ①
 a rr:TriplesMap;
  rr:logicalTable [ rr:tableName "Student" ];
  rr:subjectMap [ // ①
   rr:template "http://ex.com/Student/{ID}/{Name}"; ];
  rr:predicateObjectMap [ // ②
   rr:predicate        rdf:type;
   rr:object           foaf:Person; ];
  rr:predicateObjectMap [ // ③
   rr:predicate        rdf:type;
   rr:object           ex:Student; ];
  rr:predicateObjectMap [ // ④
   rr:predicate        ex:id ;
   rr:objectMap        [ rr:column "ID"; ] ];
  rr:predicateObjectMap [ // ⑤
   rr:predicate        foaf:name ;
   rr:objectMap        [ rr:column "Name" ] ].
```

Fig. 12. R2RML mapping of Test Case d in D007

```
1 @prefix ex: <http://ex.com/> .
2 <ex:Student/10/Venus>   foaf:name   "Venus" .
3 <ex:Student/10/Venus>   ex:id   "10"^^xsd:integer .
4 <ex:Student/10/Venus>   rdf:type   foaf:Person .
5 <ex:Student/10/Venus>   rdf:type   ex:Student .
```

Fig. 13. Result Triples from Fig. 12

```
cse:TriplesMap1Shape-ex-id // ④
 a sh:PropertyShape ;
 sh:path ex:id ;
 sh:nodeKind sh:Literal ;
 sh:datatype xsd:integer ;
 sh:minCount 1  .

cse:TriplesMap1Shape-foaf-name // ⑤
 a sh:PropertyShape ;
 sh:path foaf:name ;
 sh:nodeKind sh:Literal ;
 sh:datatype xsd:string ;
 sh:maxLength 50  .

cse:TriplesMap1Shape-rdf-type-q1 // ②
 a sh:PropertyShape ;
 sh:path rdf:type ;
 sh:nodeKind sh:IRI ;
 sh:qualifiedValueShape [ sh:in ( foaf:Person ) ] ;
 sh:qualifiedMinCount 1 ;
 sh:qualifiedMaxCount 1 ;
 sh:qualifiedValueShapesDisjoint true  .

cse:TriplesMap1Shape-rdf-type-q2  // ③
 a sh:PropertyShape ;
 sh:path rdf:type ;
 sh:nodeKind sh:IRI ;
 sh:qualifiedValueShape [ sh:in ( ex:Student ) ] ;
 sh:qualifiedMinCount 1 ;
 sh:qualifiedMaxCount 1 ;
 sh:qualifiedValueShapesDisjoint true  .

cse:TriplesMap1Shape // ①
 a sh:NodeShape ;
 sh:nodeKind sh:IRI ;
 sh:pattern "^http://ex.com/Student/(.*)/(.*)$" ;
 sh:property cse:TriplesMap1Shape-ex-id ;
 sh:property cse:TriplesMap1Shape-foaf-name ;
 sh:property cse:TriplesMap1Shape-rdf-type-q1 ;
 sh:property cse:TriplesMap1Shape-rdf-type-q2  .
```

Fig. 14. The SHACL Document Generated from Fig. 12

그림 13의 4개의 트리플은 공통된 주어 노드를 갖기 때문에 주어 노드 <ex:Student/10/Venus>를 그림 14의 node shape ①의 검증 대상으로 지정하면 4개의 트리플이 모두 검증 대상이 된다. 그림 14의 node shape의 sh:nodeKind와 sh:pattern 제약 조건에 의하면 그림 13의 공통 주어는 IRI 문자열 패턴이 그 조건에 부합함을 보인다. 또한 그림 14의 node shape은 4개의 sh:property 제약 조건에 의해 해당 주어의 트리플이 만족시켜야 하는 구조를 제시하고 있다. 그림 13의 2~5번 라인은 각각 그림

14의 ⑤, ④, ②, ③ property shape 제약 조건에 부합함으로써 그림 13의 모든 트리플은 그림 14의 SHACL 문서에 의해 설명된다. 그림 14의 ②와 ③ property shape은 Type1 PropertyShape 신택스 II로 표현되어 있다. 만약 이 둘을 Type1 PropertyShape 신택스 I로 표현했다면, 그림 13의 4~5 라인 트리플 모두 그림 14의 ②번 property shape과 ③번 property shape의 검증 대상이 된다. 그 이유는 그림 13의 4~5 라인 트리플 모두 술어가 rdf:type이기 때문이다. 따라서, 4번 라인은 ③번 property shape을 만족시키지 못하고, 5번 라인은 ②번 property shape을 만족시키지 못하게 됨으로써 검증이 실패하게 된다. 따라서, Type1 PropertyShape 신택스 II의 효능은 ②번 property shape에 부합하는 트리플은 ③번 property shape에 부합하지 않으며, ③번 property shape에 부합하는 트리플은 ②번 property shape에 부합하지 않으면 검증을 통과시킴으로써 그림 13과 같은 공유된 주어와 술어 하지만 목적어 패턴이 다른 트리플 구조를 설명할 수 있도록 하는 데 있다.

```
<TriplesMap1> // ①
 a rr:TriplesMap;
 rr:logicalTable [ rr:sqlQuery """
  SELECT Code, Name, Lan
  FROM Country
  WHERE Lan = 'EN';
 """ ] ;
 rr:subjectMap [ rr:template "http://ex.com/{Code}" ];
 rr:predicateObjectMap [
  rr:predicate  rdfs:label;
  rr:objectMap [ rr:column "Name"; rr:language "en" ]
 ].

<TriplesMap2> // ②
 a rr:TriplesMap;
 rr:logicalTable [  rr:sqlQuery """
  SELECT Code, Name, Lan
  FROM Country
  WHERE Lan = 'ES';
 """ ] ;
 rr:subjectMap [ rr:template "http://ex.com/{Code}" ];
 rr:predicateObjectMap [
  rr:predicate  rdfs:label;
  rr:objectMap [ rr:column "Name"; rr:language "es" ]
 ].
```

Fig. 15. R2RML mapping of Test Case a in D015

```
1  @prefix ex: <http://ex.com/> .
2  ex:BO rdfs:label "Bolivia, Plurinational State of"@en .
3  ex:BO rdfs:label "Estado Plurinacional de Bolivia"@es .
4  ex:IE  rdfs:label "Ireland"@en .
5  ex:IE  rdfs:label "Irlanda"@es .
```

Fig. 16. Result Triples from Fig. 15

그림 15는 D015 카테고리에 정의된 테스트 케이스 a의 R2RML 매핑 정의이고 그림 16은 이 정의에 의해 생성된 RDF 트리플들이다. 그림 17은 본 연구의 시스템이 그림 15의 R2RML 매핑 정의를 입력으로 하여 생성한 SHACL 문서이다. 그림 17에는 3개의 node shape이 생성되어 있다. node shape ①과 ②는 각각 그림 15에 정의된 2개의 triples map ①과 ②로부터 생성되었다. 2개의 triples map의 rr:template 값이 같기에 공통된 주어를 생성할 수 있으므로 Type2 NodeShape ③이 추가로 생성되었다. 그림 16에서 라인 2, 4는 triples map ①에 의해서 생성된 것이며 라인 3, 5는 triples map ②에 의해서 생성된 것이다. 그러나 라인 2, 3과 라인 4, 5는 각각 공통된 주어를 갖기 때문에, 라인 2, 3이 하나의 검증 대상 트리플 그룹을 이루며 라인 4, 5가 또 하나의 검증 대상 트리플 그룹을 이룬다. 이러한 연유로, 라인 2, 3 혹은 라인 4, 5를 node shape ①에 대입하면, 라인 3과 라인 5로 인해 검증은 실패하며 node shape ②에 대입하면 라인 2, 4로 인해 검증이 실패한다. 즉, 이 사례가 Type1 NodeShape만으로는 구조 검증이 불가능한 경우를 보여준다. 그러나 Type2 NodeShape ③에 라인 2, 3 검증을 위해 ex:Bo를

```
cse:TriplesMap1Shape-rdfs-label
 a sh:PropertyShape ;
 sh:path rdfs:label ;
 sh:nodeKind sh:Literal ;
 sh:languageIn ( "en" ) ;
 sh:maxLength 100  .

cse:TriplesMap1Shape // ①
 a sh:NodeShape ;
 sh:nodeKind sh:IRI ;
 sh:pattern "^http://ex.com/(.*)$" ;
 sh:property cse:TriplesMap1Shape-rdfs-label  .

cse:TriplesMap1_TriplesMap2_Shape // ③
 a sh:NodeShape ;
 sh:and (
  [ sh:qualifiedValueShape cse:TriplesMap1Shape ]
  [ sh:qualifiedValueShape cse:TriplesMap2Shape ]
 ) .

cse:TriplesMap2Shape-rdfs-label
 a sh:PropertyShape ;
 sh:path rdfs:label ;
 sh:nodeKind sh:Literal ;
 sh:languageIn ( "es" ) ;
 sh:maxLength 100  .

cse:TriplesMap2Shape // ②
 a sh:NodeShape ;
 sh:nodeKind sh:IRI ;
 sh:pattern "^http://ex.com/(.*)$" ;
 sh:property cse:TriplesMap2Shape-rdfs-label  .
```

Fig. 17. The SHACL Document Generated from Fig. 15

대입하면 라인 2 트리플은 node shape ①에 부합하고 라인 3은 node shape ②에 부합하기 때문에 검증은 통과한다. 같은 방식으로 Type2 NodeShape ③에 라인 4, 5 검증을 위해 ex:IE를 대입하면 라인 4 트리플은 node shape ①에 부합하고 라인 5 트리플은 node shape ②에 부합하기 때문에 검증은 통과한다. 이처럼 Type2 NodeShape은 서로 다른 triples map에 의해 생성된 트리플들이 주어 노드를 공유하게 되었을 때의 구조 검증에서 효능을 발휘한다.

V. Conclusions

RDF는 지식 그래프를 표현하기 위한 사실상의 표준이다. RDF 지식 그래프를 구축하기 위한 주요 원천 중 하나는 RDB 데이터이다. RDB 데이터로부터 RDF 지식 그래프를 구축하는 방법은 자동화된 방식으로 이루어지며 그 방식은 대중적으로 W3C의 R2RML 표준 사양을 따르고 있다. 구축된 그래프의 구조 검증은 그래프 품질 보증을 위한 필수 과정이다. 그러나 기존에는 이 단계의 작업 효율성 뿐 아니라 결과의 신뢰성 또한 보장하기 어려웠다. 근본적 이유 중 하나는 RDF 구조를 표현할 수 있는 표준화된 구문체계가 존재하지 않았기 때문이다. 이를 위해 W3C는 2017년 RDF 그래프 구조 설명 및 검증을 위한 표준 표현 포맷인 SHACL 사양을 발표하였다. 이에 SHACL 활용에 대한 다양한 후속 연구가 이루어지고 있다. 본 연구에서는 R2RML 매핑 문서를 입력으로 하여 SHACL 문서를 자동 생성하는 방법을 제안하였다. 생성된 SHACL 문서로 인해 R2RML 매핑에 의해 구축된 RDF 그래프의 구조 검증 작업을 자동화할 수 있다. 그리고 제안하는 방법으로 생성된 SHACL 문서가 검증한 결과 또한 신뢰할 수 있음을 W3C 표준 R2RML 테스트 케이스를 활용하여 수행한 실험 결과를 통해 보였다. SHACL 문서는 RDF 그래프에 대한 구조 검증 도구뿐 아니라 RDF 지식 그래프의 구조를 설명할 수 있는 도구로도 사용될 수 있다. 이에 후속 연구로서 사용자가 GUI 상에서 SHACL 신택스를 기반으로 용이하게 SPARQL 질의어를 생성케 하는 질의 도구를 개발할 계획이다.

ACKNOWLEDGEMENT

The source code of this work is hosted on https://github.com/jwchoi/Shape4RDB2RDF.

REFERENCES

[1] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," Semantic Web, Vol. 8, No. 3, pp. 489-508, December 2017. DOI: 10.3233/SW-160218

[2] Q. Xu, X. Wang, J. Li, Q. Zhang and L. Chai, "Distributed Subgraph Matching on Big Knowledge Graphs Using Pregel," IEEE Access, Vol. 7, pp. 116453-116464, August 2019. DOI: 10.1109/ACCESS.2019.2936465

[3] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," Commun. ACM, Vol. 57, No. 10, pp. 78-85, September 2014. DOI: 10.1145/2629489

[4] M. Y. Jaradeh, A. Oelen, K. E. Farfar, M. Prinz, J. D'Souza, G. Kismihók, M. Stocker, and S. Auer, "Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge," Proceedings of the 10th International Conference on Knowledge Capture, pp. 243-246, Marina Del Ray, USA, September 2019. DOI: 10.1145/3360901.3364435

[5] G. Webster, H. Nguyen, D. E. Beel, C. Mellish, C. D. Wallace, and J. Pan, "CURIOS: Connecting Community Heritage through Linked Data," Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, pp. 639-648, Vancouver, Canada, February 2015. DOI: 10.1145/2675133.2675247

[6] F. Priyatna, R. Alonso-Calvo, S. Paraiso-Medina and O. Corcho, "Querying clinical data in HL7 RIM based relational model with morph-RDB," Journal of Biomedical Semantics, Vol. 8, No. 49, October 2017. DOI: 10.1186/s13326-017-0155-8

[7] T. Liebig, A. Maisenbacher, M. Opitz, J. R. Seyler, G. Sudra and J. Wissmann, "Building a Knowledge Graph for Products and Solutions in the Automation Industry," Knowledge Graph Building Workshop Co-located with the Extended Semantic Web Conference, Portorož, Slovenia, June 2019.

[8] P. Ristoski and H. Paulheim, "RDF2Vec: RDF Graph Embeddings for Data Mining," Proceedings of The 15th International Semantic Web Conference, pp. 498-514, Kobe, Japan, September 2016. DOI: 10.1007/978-3-319-46523-4_30

[9] M. Cochez, P. Ristoski, S. P. Ponzetto and H. Paulheim, "Global RDF vector space embeddings," Proceedings of The 16th International Semantic Web Conference, pp. 190–207, Vienna, Austria, October 2017. DOI: 10.1007/978-3-319-68288-4_12

[10] V. Bellini, A. Schiavone, T. D. Noia, A. Ragone, and E. D. Sciascio, "Knowledge-aware Autoencoders for Explainable Recommender Systems," Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems, pp. 24-31, Vancouver, Canada, October 2018. DOI: 10.1145/3270323.3270327

[11] S. Vakulenko, J. D. F. Garcia, A. Polleres, M. d. Rijke, and M. Cochez, "Message Passing for Complex Question Answering over

Knowledge Graphs," Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1431-1440, Beijing, China, November 2019. DOI: 10.1145/3357384.3358026

[12] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, O. Corcho and M. Torchiano, "RDF shape induction using knowledge base profiling," Proceedings of the 33rd Annual ACM Symposium on Applied Computing, pp. 1952–1959, Pau, France, April 2018. DOI: 10.1145/3167132.3167341

[13] L. González and A. Hogan, "Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis," Proceedings of the 2018 World Wide Web Conference, pp. 1175–1184, Lyon, France, April 2018. DOI: 10.1145/3178876.3186016

[14] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," https://www.w3.org/TR/shacl/

[15] B. Spahiu, A. Maurino and M. Palmonari, "Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL," Proceedings of the 9th Workshop on Ontology Design and Patterns co-located with 17th International Semantic Web Conference, pp. 52-66, Monterey USA, October 2018. DOI: 10.3233/978-1-61499-894-5-103

[16] O. Corcho, F. Priyatna and D. Chaves-Fraga, "Towards a New Generation of Ontology Based Data Access," Semantic Web, Vol. 11, No. 1, pp. 153-160, January 2020. DOI: 10.3233/SW-190384

[17] S. Das, S. Sundara and R. Cyganiak, "R2RML: RDB to RDF Mapping Language," https://www.w3.org/TR/r2rml/

[18] D. Tarasowa, C. Lange and S. Auer, "Measuring the Quality of Relational-to-RDF Mappings," Proceedings of the 6th International Conference on Knowledge Engineering and the Semantic Web, pp. 210-224, Moscow, Russia, October 2015. DOI: 10.1007/978-3-319-24543-0_16

[19] I. Boneva, J. Dusart, D. Fernández-Álvarez and J. E. L. Gayo, "Shape Designer for ShEx and SHACL constraints," Proceedings of the 18th International Semantic Web Conference, pp. 269-272, Auckland, New Zealand, October 2019.

[20] F. J. Ekaputra and X. Lin, "SHACL4P: SHACL constraints validation within Protégé ontology editor," Proceedings of 2016 International Conference on Data and Software Engineering, pp. 1-6, Denpasar, Indonesia, October 2016. DOI: 10.1109/ICODSE.2016.7936162

[21] G. C. Publio, "SHARK: A Test-Driven Framework for Design and Evolution of Ontologies," Proceedings of the 15th Extended Semantic Web Conference, pp. 314-324, Heraklion, Greece, August 2018. DOI: 10.1007/978-3-319-98192-5_50

[22] I. Boneva, J. M. L. Aparicio and S. Staworko, "ShERML: Mapping Relational Data to RDF," Proceedings of the 18th International Semantic Web Conference, pp. 213-216, Auckland, New Zealand, October 2019.

[23] D. Beckett, T. Berners-Lee, E. Prud'hommeaux and G. Carothers, "RDF 1.1 Turtle," https://www.w3.org/TR/turtle/

[24] B. Villazón-Terrazas and M. Hausenblas, "R2RML and Direct Mapping Test Cases," https://www.w3.org/TR/rdb2rdf-test-cases/

[25] J. E. L. Gayo, H. Knublauch and D. Kontokostas, "SHACL Test Suite and Implementation Report," https://w3c.github.io/data-shapes/data-shapes-test-suite/

Authors

Ji-Woong Choi received the B.S., M.S. and Ph.D. degrees in Computer Science and Engineering from Soongsil University, Korea, in 2001, 2003 and 2011, respectively. Dr. Choi joined the faculty of the School of Computer Science and Engineering at Soongsil University, Seoul, Korea, in 2013. He is currently an Associate Professor in the School of Computer Science and Engineering, Soongsil University. He is interested in information system.