JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 9, pp. 77-89, September 2022
https://doi.org/10.9708/jksci.2022.27.09.077

Automatic Creation of SHACL Schemas for Validation of RDF Knowledge Graph Structures Based on RML Mappings

Ji-Woong Choi*

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

[Abstract]

In this paper, we propose a system which automatically generates SHACL schemas to describe and validate RDF knowledge graphs constructed by RML mappings. Unlike existing studies, the proposed system generates the schemas based on not only RML mapping rules but also metadata extracted from RML mapping input data in various formats such as CSV, JSON, XML or databases. Therefore, our schemas include the constraints on data type, string length, value range and cardinality, which were not present in the existing schemas. And we solves the problem with "repeated properties" which overlooked in existing studies. Through a conformance test consisting of 297 cases, we show that the proposed system generates correct constraints for the graphs. The proposed system can contribute to automation of the tedious and error-prone existing manual validation processes.

▸ **Key words**: RML, SHACL, RDF, Knowledge Graph, Validation

[요 약]

본 논문에서는 RML 매핑 방식으로 구축한 RDF 지식 그래프의 구조를 묘사하고 검증할 용도의 SHACL 스키마를 자동으로 생성하는 시스템을 제안한다. 제안하는 시스템은 기존 연구와는 달리 RML 매핑 규칙 뿐만 아니라 RML 매핑의 입력인 CSV, JSON, XML, 데이터베이스와 같은 다양한 포맷의 데이터에서 추출한 메타데이터도 함께 사용하여 스키마를 생성한다. 따라서 기존 연구 기반의 스키마에서는 부재했던 데이터 타입, 문자열 길이, 값의 범위, 차수 관련 제약 조건이 스키마에 포함된다. 그리고 기존 연구에서 간과한 소위 반복된 속성 문제를 제안하는 시스템은 해결한다. 297개의 케이스로 구성된 적합성 테스트를 통해 제안된 시스템이 그래프에 대한 올바른 제약 조건을 생성함을 보여준다. 제안된 시스템은 오류가 발생하기 쉬운 기존 수동 검증 프로세스를 자동화하는 데 기여할 수 있다.

▸ **주제어**: RML, SHACL, RDF, 지식 그래프, 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: 2022. 08. 23, Revised: 2022. 09. 15, Accepted: 2022. 09. 16.

Copyright © 2022 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

지식 그래프는 도메인을 구성하는 엔티티 간 관계 그리고 개별 엔티티들이 갖는 속성을 기계가 처리하기 용이한 그래프 모형으로 표현한다[1]. 최근 들어 Google, Facebook, Microsoft, Amazon, eBay 등과 같은 기업들은 대규모 지식 그래프를 적극적으로 구축하고 있다[2]. 구축의 주된 목적은 지식 그래프를 정보 검색 서비스에서 문맥 파악을 위한 도구로 사용하기 위해서다[3].

지식 그래프는 주로 다양한 형식의 기존 데이터에 일정한 규칙을 적용하여 RDF(Resource Description Framework) 형식으로 변환시키는 방식으로 구축한다[4]. 매핑 규칙을 정의하기 위해 사용하는 언어로는 RML(RDF Mapping Language)이 대표적이다[5]. RML은 현재 매핑 입력 소스 형식으로서 CSV, XML, JSON, 관계형 데이터베이스를 지원한다.

RML을 사용한 매핑 작업 시 겪는 어려움은 두 가지를 들 수 있다. 첫째, 매핑 규칙 작성자가 자신이 정의한 규칙이 생성할 그래프의 구조를 미리 확인하기 위해 참조할 적절한 대상이 없다. 이러한 어려움을 해소할 목적으로 RML 문서에 기술된 매핑 규칙을 분석하여 자동으로 그래프의 구조 정보를 생성해주는 연구 [6]이 있다. 이때 그래프의 구조는 SHACL(Shapes Constraint Language)[7]로 표현된다. SHACL은 W3C가 RDF 그래프의 구조 묘사와 검증 용도로 발표한 표준언어다. 둘째, 매핑에 의해 생성된 그래프가 의도한 구조를 준수했는지 여부를 그래프의 규모가 커질수록 검증하기 어렵다. 이러한 어려움을 해소하기 위해서는 검증기로 자동화된 그래프 구조 검증을 수행해야 하나 검증기가 사용할 스키마 작성이 선결되어야 한다. 연구 [6]의 자동 생성된 SHACL 스키마는 이 용도로 사용될 만큼 충분한 논리적 구성을 갖추지 못하고 있다.

본 논문은 RML 매핑에 의해 생성되는 그래프의 구조를 사람이 미리 파악할 수 있게 하는 용도뿐만 아니라 자동화된 그래프 구조 검증을 위해 검증기의 입력으로 사용할 용도 모두를 만족시키는 SHACL 스키마를 자동 생성하는 시스템을 제안한다. 2장에서는 기존 연구 [6]이 생성하는 그래프 스키마가 갖는 한계를 나열한다. 3장에서는 제안하는 시스템의 구조 및 처리 절차를 설명한다. 그리고 제안하는 시스템이 생성하는 SHACL 스키마가 기존 스키마의 한계를 어떻게 극복했는지 보여준다. 4장에서는 297개의 케이스로 구성된 테스트를 수행한 결과를 제시한다. 5장에서는 결론을 맺음과 동시에 향후 연구 방향을 소개한다.

II. Related Works

본 장에서는 본 논문에서 제안하는 시스템과 동일한 접근법을 사용하는 기존 연구 [6]이 생성하는 SHACL 스키마가 검증을 위해 사용될 때 발생하는 세 가지 한계를 나열한다. RML 매핑 규칙을 기반으로 SHACL 스키마를 생성하는 접근법을 사용하는 연구는 현재 [6]이 유일하며 이 연구는 RML을 고안한 조직에 의해 수행된 후속 연구다.

그림 1~4는 5장에서 수행할 테스트 셋에 포함된 한 매핑 케이스이다. 이 케이스에서 그림 1~2의 두 CSV 파일이 매핑 소스이며 그림 3이 매핑 규칙이 명세된 RML 문서이고 그림 4가 매핑 결과인 RDF 그래프이다. 그림 5는 이 케이스에 대하여 연구 [6]이 생성한 SHACL 스키마이다. 이 예는 매우 단순한 매핑 케이스이나 연구 [6]의 한계를 복합적으로 드러낼 수 있다. 참고로, 그림 3~5는 모두 RDF 표현 형식 중 하나인 Turtle[8]로 표현되어 있다. RML과 SHACL 두 언어 모두 해당 언어로 작성한 모델이면 RDF 모델이 되도록 설계하였기 때문이다.

그림 1~2에서 1행은 헤더로서 한 행이 Code와 Name이라는 컬럼으로 구성됨을 나타내며 2~3행은 각각 데이터 레코드다.

```
1: Code, Name
2: BO,   "Bolivia, Plurinational State of"
3: IE,   Ireland
```

Fig. 1. country_en.csv

```
1: Code, Name
2: BO,   "Estado Plurinacional de Bolivia"
3: IE,   Irlanda
```

Fig. 2. country_es.csv

그림 3은 두 개의 triples map을 정의하고 있다. 그림 3에서 6~17행의 TriplesMap1과 18~29행의 TriplesMap2가 triples map이다. RML에서 triples map이 매핑의 단위이다. 하나의 triples map이 하나의 트리플 패턴을 만들어내기 때문이다. tripls map은 1개의 logicalSource, 1개의 subjectMap 그리고 복수의 predicateObjectMap으로 구성된다. logicalSource는 매핑을 적용할 데이터 소스를 지정한다. subjectMap은 트리플의 주어를 생성하기 위한 매핑 규칙이고 predicateObjectMap은 그 주어에 연결될 술어와 목적어 쌍생성을 위한 매핑 규칙이다. triples map의 트리플 생성 규칙은 logicalSource의 레코드마다 적용된다. 예를 들어, 각각 1개의 술어, 목적어 쌍을 생성하는 3개의 predicateObjectMap을 갖는

triples map이 있다면 데이터 소스의 레코드 하나로부터 3개의 트리플을 생성하며 이 트리플들은 모두 주어를 공유한다. 따라서 레코드가 10개라면 결국 이 triples map은 30개의 트리플을 생성한다.

```
01: @prefix rr: <http://www.w3.org/ns/r2rml#>.
02: @prefix rml: <http://semweb.mmlab.be/ns/rml#>.
03: @prefix ql: <http://semweb.mmlab.be/ns/ql#>.
04: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
05: @base <http://example.com/base/>.

06: <TriplesMap1>
07:   a rr:TriplesMap;

08:   rml:logicalSource [
09:     rml:source "country_en.csv";
10:     rml:referenceFormulation ql:CSV ];

11:   rr:subjectMap [
12:     rr:template "http://example.com/{Code}"

13:   rr:predicateObjectMap [
14:     rr:predicate rdfs:label ;
15:     rr:objectMap [
16:       rml:reference "Name";
17:       rr:language "en" ] ].

18: <TriplesMap2>
19:   a rr:TriplesMap;

20:   rml:logicalSource [
21:     rml:source "country_es.csv";
22:     rml:referenceFormulation ql:CSV ];

23:   rr:subjectMap [
24:     rr:template "http://example.com/{Code}"];

25:   rr:predicateObjectMap [
26:     rr:predicate rdfs:label ;
27:     rr:objectMap [
28:       rml:reference "Name";
29:       rr:language "es" ] ].
```

Fig. 3. RML Graph

그림 4는 4개의 트리플이 생성된 모습이다. 2행과 5행의 IRI는 각각 주어이다. 3~4행과 6~7행의 각각의 행은 술어와 목적어 구성이다. 3행과 4행의 공통된 주어는 2행이며 6행과 7행의 공통된 주어는 5행이다. 2행과 3행의 트리플(트리플 ①)과 5행과 6행의 트리플(트리플 ②)은 TriplesMap1에 의해 생성되었으며 2행과 4행의 트리플(트리플 ③)과 5행과 7행의 트리플(트리플 ④)은 TriplesMap2에 의해 생성되었다. 이렇듯 RML 매핑은 서로 다른 triples map에 의해 생성된 트리플이 주어를 공유하는 경우가 발생한다.

```
1: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

2: <http://example.com/BO>
3:   rdfs:label "Bolivia, Plurinational State of"@en ;
4:   rdfs:label "Estado Plurinacional de Bolivia"@es.

5: <http://example.com/IE>
6:   rdfs:label "Ireland"@en ;
7:   rdfs:label "Irlanda"@es.
```

Fig. 4. Result RDF Graph by Fig. 3

```
01: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
02: @prefix sh: <http://www.w3.org/ns/shacl#>.

03: <http://example.com/base/TriplesMap1/shape>
04:   a sh:NodeShape ;
05:   sh:nodeKind sh:IRI ;
06:   sh:pattern "http://example.com/.*" ;
07:   sh:property [
08:     sh:languageIn ( "en" ) ;
09:     sh:nodeKind sh:Literal ;
10:     sh:path rdfs:label ] .

11: <http://example.com/base/TriplesMap2/shape>
12:   a sh:NodeShape ;
13:   sh:nodeKind sh:IRI ;
14:   sh:pattern "http://example.com/.*" ;
15:   sh:property [
16:     sh:languageIn ( "es" ) ;
17:     sh:nodeKind sh:Literal ;
18:     sh:path rdfs:label ] .
```

Fig. 5. Shape Graph Derived from Fig. 3

그림 5는 2개의 node shape으로 구성되어 있다. SHACL에서 node shape은 검증의 단위다. 3행~10행(node shape ①) 그리고 11행~18행(node shape ②)이 각각 node shape이다. 연구 [6]은 triples map 당 1개의 node shape을 생성한다. node shape ①은 TriplesMap1으로부터 node shape ②는 TriplesMap2로부터 유도되었다. node shape의 검증 대상 노드를 타겟 노드라 한다. RDF에서 노드는 트리플의 주어 혹은 목적어를 의미한다. 따라서 node shape은 트리플의 주어 혹은 목적어를 검증한다. node shape의 술어 sh:property의 목적어는 SHACL에서 property shape이다. property shape은 타겟 노드에 연결된 술어와 목적어 쌍 혹은 술어와 주어 쌍을 검증한다. 그림 5에서 7행~10행 그리고 15행~18행의 대괄호 부분이 property shape이다. 그림 5에 기술된 제약 조건들의 의미는 다음과 같다. 5행과 13행은 타겟 노드의 종류가 IRI여야 함을 의미한다. 6행과 14행은 타겟 노드의 IRI 문자열이 sh:pattern의 목적어인 정규식을 만족해야 한다. 10행과 18행은 타겟 노드의 술어는 rdfs:label이어야 한다. 9행과 17행은 그 술어의 목적어는 종류가 리터럴이어야 한다. 8행은 타겟 노드의 목적어 리

터럴에 “en”이라는 language tag가 있어야 한다. 16행은 8행과 비교하여 langueage tag가 “es”라는 점만 다르다.

```
1: <http://example.com/sbj1> .
2: <http://example.com/sbj2>
3:   <http://ex.com/predicate> "obj" .
```

Fig. 6. Additional Triples to Expose Vulnerability of Fig. 5

그림 6은 그림 5의 스키마가 검증에 있어서 갖는 취약성을 그림 4보다 직접적으로 드러내기 위하여 임의로 도입한 트리플들이다.

그림 6의 1행은 주어만 있고 술어와 목적어가 없는 트리플이다. 이 트리플의 주어 IRI는 그림 5의 두 node shape의 타겟 노드만을 위한 두 제약 조건 sh:nodeKind와 sh:pattern을 만족시키는 노드이다. 이 IRI를 node shape ①과 node shape ②의 타겟 노드로 설정하여 검증하면 결과는 성공이다. 그 이유는 SHACL이 property shape의 cardinality 속성에 대한 기본값이 ‘0 이상’이기 때문이다. 이처럼 기존 스키마는 그림 6의 1행 IRI처럼 property shape을 만족시키는 술어와 목적어가 없어도 검증은 성공한다. 즉, 기존 스키마의 첫 번째 문제점은 property shape에 대한 cardinality 값을 명시하지 않았다는 점이다. 이 예에서 그림 5의 property shape들의 cardinality 값은 ‘1’이어야 한다. 이 값은 매핑 규칙을 고려한 상태에서 그림 1~2에 해당하는 RML 매핑 소스 데이터의 양태를 분석해야 획득할 수 있다. 하지만 기존 스키마는 매핑 규칙에만 의존하여 생성된 것이기 때문에 cardinality를 명시하지 못한 것이다.

그림 6의 2~3행에 있는 트리플은 주어의 경우 그림 5의 두 node shape의 타겟 노드만을 위한 제약 조건은 만족시키지만 술어와 목적어의 경우 그림 5의 어느 property shape도 만족시키지 못한다. 이 트리플을 그림 5의 두 node shape에 검증 요청하면 기계는 성공이라는 결과를 출력한다. SHACL은 “open”과 “closed”라는 타겟 노드의 술어와 목적어를 검증하는 2가지 방식을 제공하며 “open”이 디폴트다. “open” 방식은 property shape의 sh:path 값과 타겟 노드의 술어가 일치하지 않으면 그 술어와 목적어는 무시된다. 즉, 오류를 유발하지 않는다. 반면 “closed” 방식은 타겟 노드가 sh:path 값으로 나열된 술어들 외의 술어를 갖는다면 오류로 판단한다. 기존 스키마는 디폴트 방식으로 검증되도록 생성되었기 때문에 그림 6의 2~3행 트리플이 검증을 통과한 것이다. 본 연구가 생성하는 SHACL 스키마의 목적은 RML 매핑 실행 전에 매핑에 의해 생성될 그래프의 구조를 예측 가능하게 함과 동시에 매핑 실행 후 생성된 그래프가 매핑 규칙에 따라 적절하게 생성되었는지를 판단하기 위함이다. 기존 스키마가 선택한 “open” 방식은 이 목적을 위해서는 적합한 선택이 아니다. 이것이 기존 스키마가 갖는 두 번째 문제점이다.

그림 4의 RDF 그래프에 대한 온전한 검증은 2행에 있는 주어 IRI를 타겟 노드로 삼아 검증을 요청하여 트리플 ①과 ③이 함께 검증된 효과를 얻고 5행에 있는 주어 IRI를 타겟 노드로 삼아 검증을 요청하여 트리플 ②와 ④가 함께 검증된 결과를 얻는 것이다. 그러나 그림 5에는 트리플 ①과 ③ 혹은 ②와 ④를 함께 검증해줄 node shape이 정의되어 있지 않다. 기존 스키마의 세 번째 문제점은 서로 다른 triples map에 의해 생성된 트리플들이 주어를 공유하는 형태로 합쳐진 경우에 대한 대비가 취약하다는 것이다. 연구 [6]의 논문은 이 경우를 위한 shape 생성 규칙을 기술하고 있다. 그러나 연구 [6]을 구현한 시스템[9]은 논문과 달리 그 규칙을 반영하지 못하고 있다. 그림 5는 연구 [6]의 구현을 실행한 결과이다. 연구 [6] 논문에 기술된 규칙이라 하더라도 그림 4처럼 합쳐진 트리플이 술어는 같으나 목적어가 다른 구조에 대한 고려가 없다. 예를 들어, 그림 5의 node shape ①에 그림 4의 2행 주어를 타겟 노드로 대입하면 그림 5의 7~10행의 property shape의 sh:path가 rdfs:label인 이유로 트리플 ①과 ③은 무조건 이 property shape의 검증대상이 되며 트리플 ①은 검증을 통과하지만 트리플 ③은 language tag가 “en”이 아니기 때문에 검증을 통과하지 못하며 이로 인해 최종적으로 검증은 실패하게 된다. 목적어에 대한 제약 조건만 다른 여러 개의 property shape을 하나의 node shape에 결합시켜야 하는 상황에서 이들을 단순히 나열하기만 한다면 늘 오류가 유발되는 구조가 된다. “repeated properties”[10]라고 불리는 이러한 구조에 대비하기 위해서는 추가적인 SHACL 구문의 구사가 요구된다.

III. The Proposed System

이 장에서는 기존 스키마의 한계를 극복함과 동시에 보다 정교한 검증을 가능케 하는 추가적인 제약 조건을 갖춘 스키마를 생성하는 본 논문이 제안하는 시스템의 구조와 처리 순서 및 방법을 기술한다.

1. System Outline

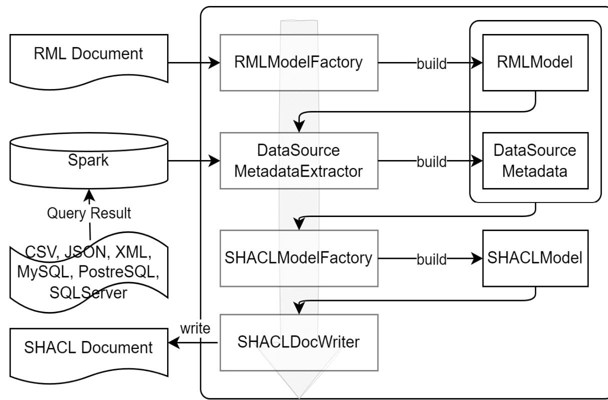

Fig. 7. Proposed System Outline

그림 7은 제안하는 시스템의 주요 구성 요소와 이들의 구동 순서를 요약한다. 이 시스템은 4단계의 처리를 거쳐 최종 출력물인 SHACL 문서를 생성한다. 각각의 단계를 담당하는 4개의 처리기가 존재한다. 처리기의 실행 순서는 RMLModelFactory, DataSourceMetadataExtractor, SHACLModelFactory, SHACLDocWriter 순이다. 1단계에서는 RML 문서를 파싱하여 RML 모델을 메모리에 구축한다. 2단계에서는 RML 매핑의 소스 데이터를 탐색 및 분석하여 SHACL 스키마에 첨가될 제약 조건 생성시 필요한 메타데이터를 추출한다. 2단계는 사용자의 선택에 따라 생략할 수 있다. 즉, 제안하는 시스템은 RML 매핑 규칙에만 의존하여 스키마를 생성할 수도 있다. 2단계를 위한 처리기는 분산 환경에서 데이터 분석이 가능한 Apache Spark라는 오픈 소스 소프트웨어에 모든 원천 데이터를 적재한 후 분석한다. 이러한 선택은 원천 데이터 볼륨이 매우 큰 상황에서도 제안하는 시스템의 안정적인 실행을 보장하기 위해서다. 그리고 RML 매핑은 다양한 포맷의 원천 데이터를 함께 연관 지어 사용할 수 있으므로 이처럼 이기종 데이터가 혼재한 상황에서도 일관된 인터페이스로 데이터 분석을 수행하기 위해서다. 3단계에서는 1~2단계에서 구축한 RML 모델과 원천 데이터의 메타데이터를 사용하여 SHACL 모델을 메모리에 구축한다. 4단계에서는 SHACL 모델 내의 shape들을 Turtle 신택스로 표현한 SHACL 문서를 최종적으로 생성한다.

2. System Details

2.1 Construction of RML Model

SHACL 모델 생성을 위한 입력 중 하나인 RML 모델은 RML 신택스를 프로그래밍 객체 모델로 왜곡 없이 옮긴 것이다. 따라서 이 모델을 사용하는 2~3단계 처리기는 RML 신택스 규칙에 따라 모델을 탐색하며 입력 RML 문서에 있던 RML 어휘의 값을 메소드 호출에 의해 조회할 수 있다. RML 어휘란 그림 3에서 rr:predicate, rml:reference처럼 "rr" 혹은 "rml" 네임스페이스 접두어가 붙은 IRI를 말하며 RDF 구문 관점에서는 모두 술어로 사용되며 의미적으로는 triples map을 구성하는 단위 매핑 규칙에 해당한다. RML은 R2RML[11]의 확장으로서 설계되었다. 즉, R2RML 문서이면 RML 문서가 되도록 설계되었다. W3C 표준인 R2RML은 RML과 같은 목적의 언어지만 입력 데이터 소스가 관계형 데이터베이스로 한정된다. 네임스페이스 접두어 "rr"이 붙은 어휘는 R2RML의 것이고 "rml"이 붙은 어휘는 RML에서 추가된 것이다.

2.2 Extraction of Data Source Metadata

SHACL 모델 생성을 위한 또 하나의 입력인 데이터 소스 메타데이터는 2단계 처리기가 구축한다. 2단계 처리기가 수행하는 일은 데이터 로딩 단계와 메타데이터 추출 단계로 나눌 수 있다. 제안하는 시스템이 Spark에 로딩시킬 수 있는 데이터 형식은 CSV, JSON, XML, 관계형 데이터베이스다. 이 중에서 CSV 형식의 경우 별도의 가공 과정 없이 triples map이 지정한 CSV 파일을 Spark에 로딩하지만 나머지 형식의 경우 해당 형식을 위한 질의어로 작성된 질의문을 실행한 결과 데이터 셋을 로딩한다. RML 사양은 데이터 형식별 질의어로서 XML의 경우 XPath, JSON의 경우 JSONPath, 데이터베이스의 경우 SQL을 지원한다. 2단계 처리기는 각각의 질의어로 작성된 질의문을 RML 모델에서 획득한다. XPath, JSONPath 질의문은 rr:iterator의 값이며 SQL 질의문은 rr:sqlQuery 혹은 rml:query의 값이다. RML에서는 입력 데이터 소스가 데이터베이스인 경우 SQL 질의문 대신 rr:tableName의 값으로 테이블명 만을 명시함으로써 입력 데이터를 한정할 수 있다. 이 경우 2단계 처리기는 RML 모델에서 획득한 테이블명을 간단히 "SELECT * FROM 테이블명" 형태의 질의문으로 변환 후 사용한다.

2단계 처리기는 로딩 단계의 마지막 처리로써 JSON과 XML에 한해 로딩된 질의 결과 데이터 셋을 평탄화한다. 그림 8은 JSON 데이터의 평탄화 연산을 요약한다. JSON과 XML은 모두 DOM(Document Object Model)을 표현하기 위한 언어이기 때문에 신택스 차이만 있을 뿐이다. 따라서 XML 데이터에 대한 평탄화 연산도 JSON 데이터에 대한 그것과 논리가 다르지 않다. 그림 8의 (ㄱ)은 질의 결과 데이터 셋의 한 행에 대응하는 객체다. (ㄴ)은 이 객

체가 Spark에 로딩된 최초 모습이며 평탄화 이전 상태다. (ㄷ)은 평탄화된 모습이다. XML과 JSON 객체는 (ㄱ)처럼 객체의 속성값으로서 객체 또는 배열을 내포한 구조가 허용되며 내포의 깊이에 대한 제한은 없다. 평탄화 연산은 객체의 속성값이 객체라면 내포된 객체의 속성명에 해당하는 새로운 컬럼을 생성한 후 내포된 객체를 분해하여 대응하는 생성된 컬럼의 값으로 할당하며 객체의 속성이 배열이라면 각각의 배열 요소들을 기존 열의 독립된 행으로 분리한다. 2단계 처리기는 배열에 객체가 내포된 구조 혹은 내포된 객체의 속성값이 배열인 구조 모두와 내포 깊이에 제한이 없는 점을 고려하여 재귀적 구조로 짜여진 논리에 의해 이 연산을 수행한다. 평탄화된 결과를 구성하는 각각의 컬럼명은 RML 문서 작성시 (ㄱ) 구조의 단위 객체에 대해서 매핑에 사용할 컬럼을 명시할 때 사용할 수 있는 모든 후보 컬럼명에 해당한다. 평탄화된 결과를 구성하는 각각의 행은 RML 매핑시 매핑 규칙이 적용되는 단위와 일치된다. 예를 들어, (ㄱ) 객체를 대상으로 RDF 목적어로서 Hobbies 컬럼이 사용되었다면 서로 다른 목적어를 갖는 2개의 트리플을 생성하기 위하여 이 객체에 매핑 규칙이 2회 적용된다. 즉, (ㄱ) 객체는 논리적 수준에서의 매핑 규칙 적용 단위이긴 하나 (ㄷ)의 행들이 실제로 매핑 규칙이 적용되는 단위이다. 따라서, 평탄화는 SHACL의 cardinality 관련 제약 조건 생성을 위해 필요한 정확한 메타데이터 산정을 위해 수행하는 구조 변경이다.

Fig. 8. Example of Flattening Operation for JSON

표 1은 2단계 처리기가 추출하는 메타데이터와 그 메타데이터가 무슨 SHACL 제약 조건의 값으로 활용되는지를 요약한다. 그리고 각각의 제약 조건이 무슨 자격을 제한하는 데 쓰이는지도 보여준다. 자격은 S(주어), P(술어), O(목적어)를 말한다.

Table 1. Extracted Metadata & Where to Be Used

Metadata	S	P	O
value type			sh:datatype
string length	sh:pattern		sh:pattern sh:minLength sh:maxLength
value range			sh:minInclusive sh:maxInclusive
cardinality		sh:minCount sh:maxCount sh:qualifiedMinCount sh:qualifiedMaxCount	

2단계 처리기는 매핑에 사용된 각각의 컬럼에 대한 값 타입을 결정한다. 값 타입은 해당 컬럼값이 목적어 자격의 RDF 리터럴로 변환될 경우 RDF 리터럴 데이터 타입으로 변환되어 sh:datatype의 값으로 사용된다. 2단계 처리기는 값 타입이 문자열 계열이면 컬럼 값의 길이 범위를 알아내고 숫자형 계열이면 컬럼 값의 범위를 알아낸다. 범위 정보 또한 해당 컬럼값이 목적어 자격의 RDF 리터럴로 변환될 경우 문자열의 최대·최소 길이 혹은 최솟값, 최댓값 표현을 위한 제약 조건의 값으로 사용된다. 문자열 길이 범위는 컬럼이 주어 혹은 목적어 자격의 IRI를 구성하는데 사용될 때도 그 IRI의 문자열 패턴을 묘사하는 sh:pattern의 값인 정규식에 쓰인다. 예를 들어, 그림 5의 6행에 있는 정규식은 그림 3의 12행에 있는 템플릿 문자열에서 유도된 것이다. 템플릿 문자열의 {Code} 부분은 매핑시 컬럼 Code의 값으로 대치된다. 제안하는 시스템은 대치되는 부분의 문자열 길이 범위를 그림 5의 6행처럼 '0 이상'의 의미인 '*' 대신 추출한 컬럼 문자열 길이 범위로 표현한다.

2단계 처리기는 property shape마다 cardinality를 명시하기 위한 메타데이터를 계산해낸다. cardinality는 주어 노드에 구조가 같은 술어·목적어 쌍이 최소·최대 몇 개 연결될 수 있는지를 표현한다. 2단계 시스템은 생성되는 트리플 패턴마다 메타데이터를 구해야 한다. 이를 위한 연산은 한 패턴의 트리플 생성을 위해 1개의 데이터 셋이 사용되는지 아니면 2개의 데이터 셋이 사용되는지에 따라 다르다. 단일 데이터 셋으로 한 패턴의 모든 트리플을 생성하는 경우는 다음의 과정을 밟는다. 우선 매핑 대상 데이터 셋을 트리플 생성에 사용된 컬럼만으로 구성된 데이터 셋으로 축소한다. 이후 중복된 행과 주어를 생성하는 컬럼들 중 일부값이 null이어서 트리플 생성이 불가능한 행을 제거한다. 그다음 주어 생성에 사용된 컬럼을 기준으로 행들을 그룹 짓는다. 마지막으로 그룹별 행수의 최솟값·최댓값을 취한다. 이 과정에서 목적어를 생성하는 컬럼들 중 일부값이 null이어서 술어·목적어 쌍 생성이 불가능한 행은 카운트에서 제외된다.

```
01: <TriplesMap1>
02:   a rr:TriplesMap;
03:   ...
04:   rr:predicateObjectMap [
05:     rr:predicate ex:practises ;
06:     rr:objectMap [
07:       a rr:RefObjectMap ;
08:       rr:parentTriplesMap <TriplesMap2>;
09:       rr:joinCondition [
10:         rr:child "Sport" ;
11:         rr:parent "ID" ; ]
12:     ]
13: ] .
```

Fig. 9. Example of Using a Referencing Object Map

그림 9는 트리플 생성에 2개의 데이터 셋이 필요한 경우를 유발하는 referencing object map 사용 사례다. 이 매핑은 TriplesMap1에 의해서 생성된 주어가 술어 ex:practises의 목적어로서 TriplesMap2에 의해서 생성된 주어를 취하는 트리플을 생성한다. 이 트리플에서 주어를 생성케 한 행의 “Sport” 컬럼값은 목적어를 생성케 한 행의 “ID” 컬럼값과 늘 같다. 이 경우는 TriplesMap1에 의해서 생성된 주어가 몇 개의 TriplesMap2에 의해서 생성된 주어와 연관될 수 있는지 그 범위를 구해야 한다. 이를 위해 우선 두 데이터 셋을 각각 주어 생성에 필요한 컬럼들과 연관 기준이 되는 컬럼으로만 구성되도록 축소한다. 이후 각각의 데이터 셋에 대하여 앞선 경우와 같은 기준으로 행 제거 연산을 수행한다. 그다음 두 데이터 셋을 조인한다. 이때 두 데이터 셋 중에서 그림 9의 TriplesMap1을 위한 데이터 셋처럼 최종적으로 주어를 생성하는 데 사용되는 데이터 셋의 행이 누락되지 않도록 하는 outer join을 사용한다. 조인 연산에 의해 하나의 데이터 셋이 만들어졌으므로 앞서 기술한 단일 데이터 셋을 사용하는 경우와 같은 방식으로 행들을 그룹짓고 그룹별 카운트의 최솟값·최댓값을 취한다. 주어 생성에 소요되는 컬럼들을 제외한 컬럼들의 값이 null인 행에 대한 카운트 역시 제외된다.

2.3 Construction of SHACL Model

SHACL 모델은 그림 10처럼 Shape 객체들의 집합으로서 설계되었다. 추상 클래스인 Shape은 NodeShape과 PropertyShape의 수퍼 클래스다. 따라서 SHACLModel이 Shape 자격으로 참조하는 객체들은 실제로는 Shape의 두 서브클래스 객체들이다. Shape, NodeShape, PropertyShape의 관계는 SHACL 사양의 정의를 따랐다.

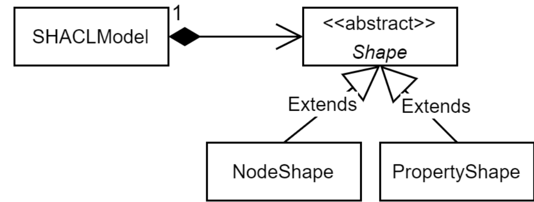

Fig. 10. UML Class Diagram of SHACL Model

표 2는 RML 어휘와 SHACL 어휘의 대응 관계를 정리한 것이다. 또한 대응된 SHACL 어휘가 어떤 자격의 트리플 요소 검증을 위해 사용되는지도 보여준다. 주어 검증을 위한 어휘의 값은 NodeShape에 저장되며 술어와 목적어 검증을 위한 어휘의 값은 PropertyShape에 저장된다. 표 2 마지막 줄의 rr:column은 데이터 셋의 컬럼명을 값으로 한다. rr:column의 값은 2단계 처리기에 의해 메타데이터 수집을 위한 대상이 되며 수집된 메타데이터는 표 1에서 정리한 대로 여러 SHACL 어휘의 값으로 사용된다.

Table 2. Terms Mapping between RML and SHACL

RML	SHACL	S	P	O
rr:termType	sh:nodeKind	✓		✓
rr:class	sh:class	✓		✓
rr:template	sh:pattern	✓		✓
rr:constant	sh:hasValue	✓		✓
rr:constant	sh:path		✓	
rr:datatype	sh:datatype			✓
rr:language	sh:languageIn			✓
rr:subject	sh:hasValue	✓		
rr:predicate	sh:path		✓	
rr:object	sh:hasValue			✓
rr:parentTriplesMap	sh:node			✓
rr:column	Refer to Table 1.			

그림 11은 3단계 처리기가 SHACL 모델을 구성하는 NodeShape과 PropertyShape 객체를 생성하는 순서를 요약한다. 그림 11의 NodeShape에 표시된 Type1~Type3의 타입 구분은 본 논문에서의 기술 편의를 위해 부여한 것이다. 세 타입 모두 그림 10의 NodeShape 객체를 생성한다. 3단계는 Type1 NodeShape 객체 생성으로 시작된다. 그다음 Type2~3 NodeShape 객체 생성을 위한 준비 작업을 수행한다. 그 후에 ProperyShape 객체들을 생성하고 이들을 Type1 NodeShape 객체에 연결함으로써 Type1 NodeShape 객체의 구성은 완료된다. 구성 완료된 Type1 NodeShape 객체를 기반으로 Type2 NodeShape 객체를 생성하고 Type1~2 NodeShape 객체를 기반으로 Type3 NodeShape 객체를 생성함으로써 3단계는 마무리된다.

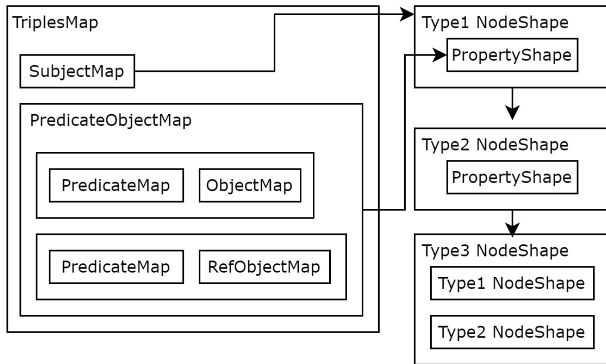

Fig. 11. Build Sequence of SHACL Model

Type1 NodeShape 객체는 RML 모델의 TriplesMap 객체마다 1개씩 존재하는 주어 노드 생성 규칙인 SubjectMap 객체에 1:1로 대응하여 생성된다. 따라서 Type1 NodeShape 객체 각각은 대응하는 triples map에 의하여 생성된 주어 노드 검증을 담당한다.

RML 매핑에서는 서로 다른 triples map에 의해 생성된 트리플들이 주어 노드를 공유하는 구조를 형성할 수 있다. Type2 NodeShape 객체는 이러한 구조의 트리플 검증을 담당할 용도로 생성한다. 더 나아가 RML 매핑에서는 공유된 주어 노드가 동시에 또 다른 트리플의 목적어 노드로도 사용될 수 있다. Type3 NodeShape 객체는 이와 같은 성격의 목적어 노드 검증을 담당하기 위하여 생성한다.

Type2~3 NodeShape 객체 생성을 위한 준비 작업은 동일한 주어 노드를 생성할 가능성이 있는 triples map들끼리 그룹을 지은 결과와 동등한 의미가 되도록 대응하는 Type1 NodeShape 객체끼리 그룹을 짓는 작업과 이 그룹화 결과를 바탕으로 향후 생성할 Type3 NodeShape 객체들의 id만 미리 생성해 두는 작업으로 구성된다. 같은 그룹에 속하게 되는 임의의 두 Type1 NodeShape 객체는 다음의 4가지 조건을 만족한다. 첫째, 두 객체의 sh:nodeKind 속성값이 같다. 둘째, 두 객체 모두 sh:pattern 속성값이 존재할 경우 sh:pattern 속성값인 정규식이 컬럼값으로 대치될 부분을 제외하고 동일하다. 셋째, 두 객체 모두 sh:hasValue 속성값이 존재할 경우 그 속성값이 같다. 넷째, 한 객체는 sh:pattern 속성값을 갖고 나머지 객체는 sh:hasValue 속성값을 가진 경우 sh:hasValue 속성값이 sh:pattern 속성값인 정규식에 매치된다. 예를 들어, 4개의 Type1 NodeShape 객체 s_1~s_4를 그룹화한 결과가 g_1={s_1, s_2, s_3}, g_2={s_4}라 하자. 그룹화에 의해 그룹 g_1처럼 멤버가 복수인 그룹이 형성되면 그 그룹의 멤버들은 동일한 주어 노드를 생성할 수 있는 triples map들에 대응하는 Typle1 NodeShape 객체들로 판정할 수 있다. 그룹화가 완료되면 멤버가 복수인 그룹에 속한 Type1 NodeSahpe 객체 각각에 대응하여 하나씩 생성할 Type3 NodeShape 객체의 id를 생성해 둠으로써 Type2~3 NodeSahep 객체 생성 준비를 끝낸다.

ProperyShape 객체는 술어·목적어 쌍 매핑 규칙과 1:1로 대응하여 생성한다. ProperyShape 객체 생성 단계에서는 “repeated properties” 구조에 대비하기 위한 작업이 포함된다. 이 작업은 한 TriplesMap 객체에 속한 술어·목적어 쌍 매핑 규칙들 중 술어 매핑 규칙이 동일한 것들이 발견되면 해당 술어·목적어 쌍 매핑 규칙으로부터 생성되는 PropertyShape 객체의 한 속성으로서 술어 제약 조건이 중복됨을 기록해 둔다. 이 속성에 의해 향후 출력될 property shape의 SHACL 구문 구성을 달라한다. 또한 ProperyShape 객체 생성 단계에서는 공유된 주어가 목적어로 사용되는 구조에 대한 대비도 동반된다. 이 구조는 그림 9에서처럼 목적어 생성을 위해 referencing object map이 사용된 경우에 발생할 수 있다. referencing object map을 구성하는 매핑 규칙 중 하나인 rr:parentTriplesMap의 값인 triples map 이름은 sh:node 제약 조건의 값 생성에 사용된다. 그 triples map이 다른 triples map과 동일한 주어 노드를 생성할 가능성이 없는 것이라면 sh:node의 값은 그 triples map에 대응하여 생성한 Type1 NodeShape 객체의 id가 된다. 그러나 그렇지 않다면 Type1 NodeShape 객체의 id 대신 그 Type1 NodeShape 객체에 대응하여 향후에 생성할 Type3 NodeShape 객체의 id를 sh:node의 값으로 할당한다. 이 사용을 위해 Type3 NodeShape 객체의 id만 미리 준비해 둔 것이다. 이러한 다양한 고려를 반영하여 생성한 PropertyShape 객체 각각은 자신을 생성케 한 술어·목적어 쌍 매핑 규칙이 속한 TriplesMap 객체에서 생성된 Type1 NodeShape 객체에 자신의 id를 등록시킨다. 등록된 id는 Type1 NodeShape 객체의 sh:property 속성값으로 쓰인다. 이로써 Type1 NodeShape 객체는 타겟 노드의 술어와 목적어도 검증할 수 있는 구조를 완성한 것이다.

Type2 NodeShape 객체는 앞서 구해 놓은 Type1 NodeShape 객체들의 그룹을 사용하여 생성한다. Type2 NodeShape 객체는 멤버가 복수인 그룹의 멤버간 조합에 대응되어 생성된다. 앞선 예의 그룹 g_1에서 만들어지는 조합은 {s_1, s_2}, {s_1, s_3}, {s_2, s_3}, {s_1, s_2, s_3} 이다. 각각의 조합은 주어 노드가 공유될 수 있는 각각의 경우에 해당한다. 따라서 각각의 Type2 NodeShape 객체는 대응하는 조합의 멤버들이 보유한 제약 조건을 결합한 의미를 갖는다. Type2 NodeShape 객체는 주어 노드 검증 목적의 제

약 조건으로만 구성된 형태로 우선 생성된다. 주어 노드 검증용 제약 조건은 sh:nodeKind, sh:pattern, sh:hasValue, sh:class이다. sh:nodeKind는 조합 멤버 모두가 같은 값을 갖기 때문에 Type2 NodeShape 객체도 그 값을 그대로 취한다. sh:hasValue은 조합 멤버 모두는 아니더라도 이 제약 조건을 갖는 것들은 모두 같은 값을 갖기 때문에 Type2 NodeShape 객체도 그 값을 그대로 취한다. 다만 조합 멤버 중 sh:pattern을 보유한 것이 있다면 sh:pattern의 정규식에 sh:hasValue 값이 매치되기 때문에 이 경우 Type2 NodeShape 객체는 sh:hasValue는 취하지 않고 sh:pattern만 취한다. sh:pattern의 경우 이 제약 조건을 갖는 멤버들은 컬럼값으로 대치되는 부분을 제외하고 동일한 정규식을 갖는다. 컬럼값으로 대치되는 부분에는 대치될 문자열의 길이 범위가 표현되어 있다. 그러나 이 범위는 멤버별로 컬럼을 다르게 지정했을 수 있으므로 제각각일 수 있다. 따라서 Type2 NodeShape 객체는 제각각의 범위 중 가장 낮은 하한과 가장 높은 상한으로 새롭게 구성한 범위를 취한다. sh:class의 경우 조합 멤버들이 갖는 이 제약 조건의 값을 모두 취한다. 주어 노드 검증 목적의 제약 조건 구성이 완료되면 Type2 NodeShape 객체는 대응하는 조합의 멤버가 소유한 모든 PropertyShape 객체들을 자신의 것으로 취한다. 이 과정에서도 "repeated properties" 구조에 대한 대비를 수행한다. 만약 자신이 속해있던 Type1 NodeShape 객체에서는 다른 PropertyShape 객체와 제약하는 술어가 중복되지 않았던 PropertyShape 객체가 다른 Type1 NodeShape 객체에서 온 PropertyShape 객체와 제약하는 술어가 중복된다면 이 PropertyShape 객체와 동일한 제약 조건을 갖는 새로운 PropertyShape 객체를 생성한다. 그리고 제약하는 술어가 중복됨을 새로 생성한 PropertyShape 객체에 추가적으로 기록한다. 그러고 나서 Type2 NodeShape 객체는 새로 생성한 것을 Type1 NodeShape 객체에서 가져온 것 대신 자신의 것으로 취한다.

Type3 NodeShape 객체는 앞서 구해 놓은 Type1 NodeShape 객체들의 그룹 중 멤버가 복수인 그룹의 멤버 각각에 대응하여 하나씩 생성된다. Type3 NodeShape에 대해서는 앞선 예의 그룹 g_1에 속한 s_1을 대표로 설명하고자 한다. s_1에 대응하여 생성된 Type3 NodeShape 객체는 s_1의 id와 그룹 g_1의 멤버간 조합 중 s_1이 포함된 조합 즉, $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_1, s_2, s_3\}$에 대응시켜 생성한 Type2 NodeShape 객체들의 id를 원소로 하는 집합이 유일한 속성이다. Type3 NodeShape 객체는 다음 단계에서 이 Type1과 Type2 NodeShape 객체들의 id를 OR 연산자로 결합시킨 구조의 SHACL 구문을 출력한다. 그 이유는 s_1에 대응하는 tripes map에 의해 생성된 트리플이 다른 트리플과 주어를 공유하는 형태로 결합될 수 있는 구조는 결국 이 id들이 설명하는 구조 중 하나가 되기 때문이다.

2.4 SHACL Serialization

마지막 4단계는 SHACL 모델에 등록된 Shape 객체 각각에 대하여 그 Shape 객체가 보유한 속성값들을 SHACL 신택스에 따라 구성한 문자열을 반환하는 함수를 호출한다. 호출 결과 얻어진 문자열들을 파일에 기록함으로써 SHACL 문서 생성은 완료된다.

그림 12~15는 그림 3의 RML 문서로부터 제안하는 시스템이 생성한 하나의 SHACL 문서를 설명하고자 하는 의미 단위로 분절하여 제시한 것이다.

그림 12는 그림 3의 TriplesMap1으로부터 생성한 Type1 NodeShape 객체와 PropertyShape 객체가 출력한 SHACL 구문이다. 5~11행의 node shape는 sh:property의 값으로 12~20행의 property shape를 참조하고 있다. 7행의 sh:closed는 값이 true다. 이 조건으로 인해 해당 node shape은 타겟 노드의 술어·목적어 쌍 검증을 "closed" 방식으로 수행한다. 제안하는 시스템은 Type1과 Type2 NodeShape 객체가 출력한 node shape에 이 조건을 항상 포함시킨다. 10행 정규식 내의 컬럼값으로 치환될 부분의 최소 문자열 길이인 2와 17~20행의 조건값들은 모두 데이터 소스에서 수집한 메타데이터로부터 수집된 것들이다.

```
01: @prefix my: <http://my.example/ns#>.
02: @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
03: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
04: @prefix sh: <http://www.w3.org/ns/shacl#>.

05: my:TriplesMap1Shape
06:     a sh:NodeShape ;
07:     sh:closed true ;
08:     sh:ignoredProperties (rdf:type) ;
09:     sh:nodeKind sh:IRI ;
10:     sh:pattern "^http://example.com/(.{2,})$" ;
11:     sh:property my:TriplesMap1Shape-rdfs-label .

12: my:TriplesMap1Shape-rdfs-label
13:     a sh:PropertyShape ;
14:     sh:path rdfs:label ;
15:     sh:nodeKind sh:Literal ;
16:     sh:languageIn ( "en" ) ;
17:     sh:minLength 7 ;
18:     sh:maxLength 31 ;
19:     sh:minCount 1 ;
20:     sh:maxCount 1 .
```

Fig. 12. Part 1 of Our Shape Graph Derived from Fig. 3

```
01: my:TriplesMap2Shape
02:     a sh:NodeShape ;
03:     sh:closed true ;
04:     sh:ignoredProperties (rdf:type) ;
05:     sh:nodeKind sh:IRI ;
06:     sh:pattern "^http://example.com/(.{2,})$" ;
07:     sh:property my:TriplesMap2Shape-rdfs-label .

08: my:TriplesMap2Shape-rdfs-label
09:     a sh:PropertyShape ;
10:     sh:path rdfs:label ;
11:     sh:nodeKind sh:Literal ;
12:     sh:languageIn ( "es" ) ;
13:     sh:minLength 7 ;
14:     sh:maxLength 31 ;
15:     sh:minCount 1 ;
16:     sh:maxCount 1 .
```

Fig. 13. Part 2 of Our Shape Graph Derived from Fig. 3

그림 13은 그림 3의 TriplesMap2로부터 생성한 Type1 NodeShape 객체와 PropertyShape 객체가 출력한 SHACL 구문이다. TriplesMap1과 TriplesMap2가 구조가 같기 때문에 그림 12와 그림 13 또한 구조가 동일하다. 두 그림 모두 데이터 소스 메타데이터에서 가져온 값도 같음을 확인할 수 있다. 그러나 이것은 두 triples map이 사용하는 CSV 파일이 각자 다름에도 불구하고 우연히 두 데이터 셋의 양태가 같은 결과에 기인한 것이다.

그림 14의 1~8행에 있는 node shape은 Type2 NodeShape의 출력 사례에 해당한다. 이 node shape은 그림 12와 그림 13의 두 node shape이 sh:nodeKind와 sh:pattern의 값이 같은 이유로 생성된 것이다. 이 node shape이 참조하는 그림 14에 있는 2개의 property shape은 모두 "repeated properties" 구조에 대한 대비가 반영된 사례다. 그림 12의 property shape과 그림 13의 property shape이 제약하는 술어가 둘 다 rdfs:label로 중복되기 때문이다. 그림 14의 두 property shape을 관찰해 보면 기존 property shape의 목적어 노드 검증을 위한 제약 조건들을 통째로 sh:qualifiedValueShape의 값으로 취하고 있음을 확인할 수 있다. 또한 기존 property shape의 sh:minCount와 sh:maxCount 대신 sh:qualifiedMinCount와 sh:qualifiedMaxCount가 사용되었음을 확인할 수 있다. SHACL에서 이러한 구조를 사용하면 타겟 노드가 술어는 같지만 목적어의 구조가 상이한 여러 술어·목적어 쌍을 가지고 있을 때 술어가 중복됨에도 불구하고 목적어의 구조별로 각각 독립적으로 검증할 수 있게 된다.

```
01: my:TriplesMap1-TriplesMap2ShapeAnd
02:     a sh:NodeShape ;
03:     sh:closed true ;
04:     sh:ignoredProperties (rdf:type) ;
05:     sh:nodeKind sh:IRI ;
06:     sh:pattern "^http://example.com/(.{2,})$" ;
07:     sh:property my:TriplesMap1Shape-rdfs-label-q1 ;
08:     sh:property my:TriplesMap2Shape-rdfs-label-q1 .

09: my:TriplesMap1Shape-rdfs-label-q1
10:     a sh:PropertyShape ;
11:     sh:path rdfs:label ;
12:     sh:qualifiedValueShape [
13:         sh:nodeKind sh:Literal ;
14:         sh:languageIn ( "en" ) ;
15:         sh:minLength 7 ;
16:         sh:maxLength 31
17:     ] ;
18:     sh:qualifiedMinCount 1 ;
19:     sh:qualifiedMaxCount 1 .

20: my:TriplesMap2Shape-rdfs-label-q1
21:     a sh:PropertyShape ;
22:     sh:path rdfs:label ;
23:     sh:qualifiedValueShape [
24:         sh:nodeKind sh:Literal ;
25:         sh:languageIn ( "es" ) ;
26:         sh:minLength 7 ;
27:         sh:maxLength 31
28:     ] ;
29:     sh:qualifiedMinCount 1 ;
30:     sh:qualifiedMaxCount 1 .
```

Fig. 14. Part 3 of Our Shape Graph Derived from Fig. 3

그림 15의 두 node shape은 이 SHACL 문서의 다른 shape에 의해 참조되고 있지는 않지만 Type3 NodeShape의 출력 사례에 해당한다. 두 node shape은 그림 12~14에 있는 node shape들의 id를 sh:or로 연결한 모습을 확인할 수 있다. 1~6행의 node shape은 TriplesMap1에 의해 생성된 주어 노드가 타겟 노드일 때 그리고 7~12행의 그것은 TriplesMap2에 의해 생성된 주어 노드가 타겟 노드일 때 그 타겟 노드가 취할 수 있는 구조들의 모든 후보를 제시한 것이다.

```
01: my:TriplesMap1ShapeOr
02:     a sh:NodeShape ;
03:     sh:or (
04:         my:TriplesMap1-TriplesMap2ShapeAnd
05:         my:TriplesMap1Shape
06:     ) .

07: my:TriplesMap2ShapeOr
08:     a sh:NodeShape ;
09:     sh:or (
10:         my:TriplesMap1-TriplesMap2ShapeAnd
11:         my:TriplesMap2Shape
12:     ) .
```

Fig. 15. Part 4 of Our Shape Graph Derived from Fig. 3

그림 4의 2행에 있는 주어 노드를 그림 12의 node shape로 검증하면 그림 4의 4행 술어·목적어 쌍이 sh:languageIn 제약 조건을 만족시키지 못하기 때문에 실패한다. 마찬가지로, 그림 4의 5행의 주어 노드를 그림 13의 node shape로 검증하면 그림 4의 6행 술어·목적어 쌍이 sh:languageIn 제약 조건을 만족시키지 못하기 때문에 실패한다. 그러나 이 두 주어 노드를 그림 14의 node shape로 검증하면 모두 성공한다. 둘 다 구조가 같기 때문이다. 즉, 두 주어 노드 모두 술어가 rdfs:label인 술어·목적어 쌍 2개씩을 가지고 있으며 그중 하나는 TriplesMap1에 의해 생성된 것이고 나머지 하나는 TriplesMap2에 의해 생성된 것이다.

기존 연구 [6]이 생성하는 SHACL 스키마가 검증 용도로 사용하고자 할 때 도출된 3가지 한계를 제안하는 시스템이 생성하는 SHACL 스키마가 모두 극복한 한 사례를 보였다. 첫째, 모든 property shape에 cardinality가 부여된 것을 확인할 수 있다. 둘째, “closed” 검증 방식을 취했음을 확인할 수 있다. 셋째, “repeated properties” 구조에 대한 대비가 마련되어 있음을 확인할 수 있다.

IV. Conformance Test

제안하는 시스템이 생성하는 SHACL 스키마가 범용적으로 사용하기에 어느 정도 적합한지를 평가하고자 한다. 이러한 목적으로 개발된 테스트 케이스 집합은 아직 존재하지 않기 때문에 본 연구에서는 RML 프로세서의 정확성을 평가하고자 개발된 테스트 케이스 집합 [12]를 대용하고자 한다. RML 프로세서는 RML 매핑 규칙에 따라 RDF 그래프를 생성시키는 시스템을 의미한다. [12]의 각각의 테스트 케이스는 데이터 셋, RML 문서, RDF 문서로 구성되어 있다. 데이터 셋과 RML 문서는 RML 프로세서의 입력이며 RDF 문서는 RML 프로세서가 출력해야 할 정답이다. 본 평가에서는 데이터 셋과 RML 문서를 제안하는 시스템의 입력으로 사용하며 RDF 문서를 제안하는 시스템이 생성한 SHACL 스키마가 검증할 RDF 그래프로 사용한다. 테스트 케이스별로 SHACL 스키마와 RDF 문서를 검증기에 입력한 후 모든 트리플에 대해 오류가 없다는 검증 결과가 출력되면 해당 테스트 케이스는 성공으로 판단한다. 이것은 SHACL 스키마가 RDF 문서의 트리플이 형성하고 있는 모든 구조를 빠짐없이 묘사하고 있다는 의미이기 때문이다.

[12]는 60개 카테고리로 분류된 총 297개의 테스트 케이스로 구성되어 있다. 카테고리별로 RML 문서의 매핑 규칙 구성을 달리한다. 같은 카테고리로 묶인 테스트 케이스들은 입력 데이터 소스 종류만 달리한다. 사용된 입력 데이터 소스는 CSV, XML, JSON 형식의 파일과 오픈소스 RDBMS인 MySQL, PostgreSQL 그리고 Microsoft의 RDBMS인 SQLServer가 있다. 카테고리마다 모든 입력 데이터 소스에 대한 테스트 케이스가 정의된 것은 아니다. 본 평가는 각각의 테스트 케이스가 정한 입력 데이터 소스를 그대로 사용했으며 모든 테스트 케이스를 평가에 사용하였다. 아래 표 3은 평가 결과를 요약한다. 평가 결과는 4가지 유형으로 분류된다.

Table 3. Test Summary

Result	# of categories
schema creation failed	4
schema created but no graph	8
schema created but validation failed	6
validation success	42

첫 번째는 SHACL 스키마 생성에 실패한 경우로서 4개의 카테고리에서 발생했다. 이 4개의 카테고리는 테스트 사양에서 RML 문서에 구문 오류를 의도적으로 심어놓고 RML 프로세서가 그래프 생성에 실패해야 테스트를 통과한 것으로 판정하고자 설계한 것이다. 이 경우 제안하는 시스템은 RML 모델 생성 단계에서 RML 문서의 구문 오류를 인지하며 해당 원인을 출력하고 처리를 종료하는 방식으로 동작한다.

두 번째는 SHACL 스키마 생성에 성공했으나 검증할 그래프가 제공되지 않은 경우로서 8개의 카테고리에서 발생했다. 이 8개의 카테고리 또한 테스트 사양에서 의도적으로 존재하지 않거나 빈 데이터 셋으로의 접근 혹은 잘못된 질의문의 사용 아니면 존재하지 않는 컬럼으로의 접근을 유발하여 RML 프로세서가 그래프 생성에 실패하도록 만든 테스트 케이스들이다. 이 경우 제안하는 시스템은 데이터 소스 메타데이터 수집 단계에서 데이터 셋의 잘못된 접근 혹은 접근을 인지한 후 메타데이터 사용 없이 RML 매핑 규칙만을 사용하여 SHACL 스키마를 생성했다.

세 번째는 SHACL 스키마 생성에 성공했으나 주어진 그래프를 성공적으로 검증하지 못한 경우로서 6개의 카테고리에서 발생했다. 이 6개의 카테고리에 속한 RML 문서는 모두 RDF 그래프가 N-Quads 신택스[13]로 생성되도록 설계되었다. N-Quads에서 한 문장은 주어, 술어, 목적어, 그래프 IRI 이렇게 4개의 요소로 구성된다. 그래프 IRI는 트리플이 속한 그래프 이름을 의미한다. 즉, N-Quads

는 트리플과 해당 트리플의 소속 그래프명을 한 문장에 표현할 수 있도록 한 신택스다. 제안하는 시스템이 N-Quads로 표현된 그래프를 검증할 수 없는 이유는 SHACL에는 트리플이 특정 그래프에 속해야 한다고 제한할 수 있는 제약 조건이 존재하지 않기 때문이다.

나머지 모든 카테고리의 테스트 케이스에 대해서는 제안하는 시스템이 생성한 SHACL 스키마로 주어진 그래프의 모든 트리플을 성공적으로 검증하였다. 결과적으로 제안하는 시스템은 의도적으로 오류를 유발한 경우와 SHACL 사양의 한계로 인하여 검증이 애초에 불가능한 경우를 제외하면 모든 테스트 케이스에 대한 정확한 SHACL 스키마를 생성했다고 평가할 수 있다.

V. Conclusions

본 논문을 통해 RML 매핑에 의해 생성되는 그래프의 구조를 사람이 미리 파악할 수 있게 하는 용도뿐만 아니라 자동화된 그래프 구조 검증을 위해 검증기의 입력으로 사용할 용도 모두를 만족시키는 SHACL 스키마를 자동 생성하는 시스템의 구조와 처리 방식을 소개했다. 동일 목적의 기존 연구에 기반한 SHACL 스키마가 그래프 검증 용도로 사용될 때 드러내는 한계를 제안하는 시스템이 생성한 SHACL 스키마로 극복할 수 있음을 동일 사례 분석을 통해 보였을 뿐만 아니라 다양한 케이스의 테스트를 통해 본 논문의 구현이 범용적으로 사용될 수 있음을 보였다. 근래 들어 RML로 생성하는 RDF 지식 그래프의 규모가 대형화되는 추세다. 연구 [14-16]은 각각 소위 빅 데이터의 속성을 만족하는 소셜 미디어 데이터, IoT 디바이스에서 수집된 헬스케어 데이터, 의생명 데이터를 근간으로 RML을 사용하여 대규모 RDF 그래프를 구축한 최근 사례들이다. 본 연구의 산물은 이처럼 지식 그래프의 규모가 커질수록 그 생성 과정에서 더욱 두드러지는 기존 비효율적인 수동 검증 프로세스를 자동화하는데 기여할 수 있다.

ACKNOWLEDGEMENT

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

REFERENCES

[1] H. Paulheim, "Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods," Semantic Web, Vol. 8, No. 3, pp. 489-508, December 2016. DOI: 10.3233/SW-160218

[2] L. Bellomarini, E. Sallinger, and S. Vahdati, "Knowledge Graphs: The Layered Perspective," Knowledge Graphs and Big Data Processing, Vol. 12072, No. 2, pp. 20-34, July 2020. DOI: 10.1007/978-3-030-53199-7_2

[3] R. Reinanda, E. Meij, and M. d. Rijke, "Knowledge Graphs: An Information Retrieval Perspective," Foundations and Trends® in Information Retrieval, Vol. 14, No. 4, pp. 289-444, October 2020. DOI: 10.1561/1500000063

[4] A. Dimou, "High-quality knowledge graphs generation: R2rml and rml comparison, rules validation and inconsistency resolution," Applications and Practices in Ontology Design, Extraction, and Reasoning, Vol. 49, No. 4, pp. 55-72, November 2020. DOI: 10.3233/SSW200035

[5] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, and M. Vida, "SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs," Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 3039-3046, Virtual Event, Ireland, October, 2020. DOI: 10.1145/3340531.3412881

[6] T. Delva, B. D. Smedt, S. M. Oo, D. V. Assche, S. Lieber, and A. Dimou, "RML2SHACL: RDF Generation Is Shaping Up," Proceedings of the 11th on Knowledge Capture Conference (K-CAP '21), pp. 153-160, New York, USA, December 2021. DOI: 10.1145/3460210.3493562Q

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

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

[9] T. Delva, and S. M. Oo, "RML2SHACL," https://github.com/RMLio/RML2SHACL/

[10] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," Springer Nature, pp. 251-253, 2022.

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

[12] P. Heyvaert, A. Dimou and B. D. Meester, "RML Test Cases," https://rml.io/test-cases/

[13] G. Carothers, "RDF 1.1 N-Quads," https://www.w3.org/TR/n-quads/

[14] B. Abu-Salih, M. Al-Tawil, I. Aljarah, H. Faris, P. Wongthongtham, K. Y. Chan, and A. Beheshti, "Relational Learning Analysis of Social Politics using Knowledge Graph Embedding," Data Mining and Knowledge Discovery, Vol. 35, No. 4, pp. 1497-1536, July 2021. DOI: 10.1007/s10618-021-00760-w

[15] R. Reda, F. Piccinini, and A. Carbonaro, "Semantic Modelling of Smart Healthcare Data," Proceedings of SAI Intelligent Systems Conference, pp. 399-411, London, United Kingdom, September 2018. DOI: 10.1007/978-3-030-01057-7_32

[16] G. Alor-Hernández, J. L. Sánchez-Cervantes, A. Rodríguez-González, and R. Valencia-García, "Current Trends in Semantic Web Technologies: Theory and Practice," Springer, pp. 25-56, 2019.

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 Data and Knowledge, Artificial Intelligence and Machine Learning.