JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 11, pp. 229-241, November 2024
https://doi.org/10.9708/jksci.2024.29.11.229

A Common Shape Model To Obtain Multiple Schema Representations For RDF Knowledge Graphs Derived From 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 to provide schemas written in multiple languages for RDF knowledge graphs derived from RML mappings. The core of the system is a common shape model designed to support both SHACL and ShEx, which are all languages for representing the structure of RDF graphs, called “shapes”. The model is syntax-neutral, but includes enough data to represent constraints with the two schema languages. The data in the model are automatically collected from RML mapping rules themselves and metadata about input data sources in RML mappings. Also, the presented system provides a common interface to support translation of the model to multiple languages. The translation to each language is performed by each syntax-specific implementation of the interface, so the model data are rearranged according to each language’s syntax. This paper describes the proposed method in detail and also presents implementation results.

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

[요 약]

본 논문에서는 RML 매핑으로 유도된 RDF 지식 그래프에 대한 스키마를 공통 형상 모델이라는 구조를 통해 여러 언어로 자동 생성해 주는 시스템을 제안한다. 형상이란 그래프 스키마 언어에서 그래프가 갖추어야 할 제약 조건의 집합을 뜻한다. 공통 형상 모델의 구조는 특정 스키마 언어의 구문에 독립적이다. 이 모델을 구성하는 데이터는 사용자가 RML 문법에 따라 정의한 매핑 규칙 그리고 RML 매핑에서 입력 자격인 다양한 포맷의 원천 데이터를 정해진 알고리즘에 따라 분석 및 가공 후 수집된 것들이다. 제안하는 시스템은 개별 스키마 언어들의 구문에 맞게 모델 데이터를 배열하는 방법을 정의할 수 있게 하는 공통 인터페이스를 제공한다. 즉, 개별 언어는 이 인터페이스를 구현하기만 하면 자신의 구문으로 스키마를 출력할 수 있다. 본 논문에서는 제안하는 방법의 실용성을 보이기 위해 대표적인 RDF 그래프 스키마 언어인 SHACL과 ShEx로 스키마를 출력하도록 구현하였으며 출력 스키마가 유효함을 보이는 테스트 결과를 제시한다.

▸ **주제어**: RML, SHACL, ShEx, RDF, 지식 그래프

• 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: 2024. 10. 17, Revised: 2024. 11. 04, Accepted: 2024. 11. 06.

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

I. Introduction

지식 그래프는 도메인을 구성하는 개념 혹은 사실들의 관계를 그래프 모형으로 구조화한 것이다[1]. 최근 들어 챗봇, 검색, 추천 등과 같은 AI 기반 서비스에서 도메인 식별, 도메인 내 존재하는 사실 혹은 문맥 정보 취득을 목적으로 지식 그래프를 훈련 데이터로 사용하거나 지식 그래프 자체에 대한 임베딩을 획득하여 사용하고 있다[2].

지식 그래프는 전통적으로 W3C 표준인 RDF(Resource Description Framework) 포맷으로 구축해 왔다[3]. 대규모 RDF 지식 그래프는 주로 데이터베이스나 CSV, JSON, XML 형식의 데이터에 매핑 규칙을 적용하여 RDF 요소로 일괄적으로 변환시키는 방식으로 구축된다[4]. RML(RDF Mapping Language)은 이러한 방식에서 사실상 표준으로 사용되는 매핑 규칙 정의 언어다[5].

본 논문에서는 RML 매핑으로 생성되는 RDF 그래프에 대한 스키마를 자동 생성해 주는 시스템을 제안한다. 제안하는 시스템은 그래프 스키마를 서로 다른 구문 체계를 갖는 여러 언어로 출력할 수 있을 뿐 아니라 출력 언어가 달라지더라도 단일한 방식으로 생성할 수 있는 구조로 설계하였다. 단일한 방식의 핵심은 공통 형상 모델에 있다. 형상(shape)은 그래프 스키마 언어들에서 그래프가 갖추어야 할 제약 조건의 집합을 뜻한다. 이 모델은 특정 스키마 언어의 구문에 독립적인 구조를 취하되 스키마 생성을 위해 요구되는 데이터를 충분히 제공한다. 이 모델을 구성하는 데이터는 사용자에 의해 작성된 RML 매핑 규칙 그리고 매핑의 입력인 다양한 포맷의 데이터로부터 정해진 알고리즘에 따라 분석 및 가공 후 수집된 것들이다. 공통 형상 모델은 개별 스키마 언어들의 구문에 맞게 모델 데이터를 배열하는 방법을 정의할 수 있도록 하는 공통된 인터페이스를 제공한다. 즉, 개별 언어는 이 인터페이스를 구현하기만 하면 자신의 구문으로 스키마를 출력할 수 있다. 본 논문에서는 공통 형상 모델을 기반으로 대표적인 RDF 그래프 스키마 언어인 SHACL(Shapes Constraint Language)[6]과 ShEx(Shape Expression)[7]로 스키마를 출력하도록 인터페이스를 구현하였다.

제안 시스템은 RML 매핑 규칙과 매핑의 입력 자격 데이터를 함께 사용해 스키마를 생성할 수도 있지만 RML 매핑 규칙만을 사용해 스키마를 생성케 하는 옵션도 제공한다. 전자의 방식은 기계로 그래프 구조 검증이 가능한 수준의 정교한 스키마를 생성한다. 이 스키마는 RML 매핑으로 생성된 그래프의 구조 검증 용도 혹은 다른 방식으로 생성된 그래프와의 통합 시에도 그래프 구조의 일관성을 효율적으로 관리하기 위한 수단으로 사용될 수 있다. 후자의 방식은 스키마의 정교함은 떨어지지만 데이터 분석 과정이 생략되므로 빠른 스키마 생성이 장점이다. 이 방식으로 생성한 스키마는 RML 매핑 규칙 작성자가 자신이 작성한 매핑 규칙이 본래 의도한 구조의 그래프를 생성하는지를 미리 확인하고자 하는 상황에서 충분한 정보를 제공한다. 따라서 의도에 부합하는 매핑 규칙을 효율적으로 작성하는 데 도움을 줄 수 있다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 2장에서는 기존 연구들의 의의와 한계를 기술한다. 3장에서는 제안하는 시스템의 구조 및 처리 절차를 설명한다. 4장에서는 테스트를 수행한 결과를 제시한다. 5장에서는 본 논문의 내용을 요약하고 정리한다.

II. Related Works

본 장에서는 매핑 규칙에 따라 구축된 RDF 그래프를 위한 스키마를 자동 생성하는 기존 연구들을 정리한다. 그 중에서 본 연구와 같이 RML 매핑 규칙으로부터 그래프 스키마를 생성하는 기존 연구에 대해서는 그 한계를 상세히 분석하여 이를 본 연구에서 해결하고자 한다.

1. Generating Shapes From Mapping Rules

RDF 그래프를 매핑 방식으로 생성코자 할 때 사용되는 대표적인 기술은 Direct Mapping[8], R2RML(RDB to RDF Mapping Language)[9], RML이 있다. Direct Mapping과 R2RML은 W3C의 표준으로서 관계형 데이터베이스(RDB)로부터 RDF 그래프를 생성시키는 것을 목적으로 개발되었다. RML은 RDB뿐 아니라 CSV, JSON, XML 포맷 데이터로도 RDF 그래프를 생성할 수 있도록 한 R2RML에 대한 확장 사양이다. R2RML과 RML은 매핑 규칙 정의 언어라서 사용자의 요구에 맞춘 매핑 규칙의 정의가 가능하며 이때 매핑 시 입력으로 사용될 데이터의 범위 또한 설정 가능하다. 반면에 Direct Mapping은 RDB의 근간인 관계형 모델의 의미에 준하여 RDB 요소를 RDF 요소로 대응시킨 하나의 고정적인 매핑 규칙으로서 언제나 하나의 RDB 데이터 전체를 변환 대상으로 한다.

[10-13]은 Direct Mapping으로 생성된 RDF 그래프의 스키마를 자동 생성하기 위한 연구들이다. [10-11]은 SHACL 스키마를 생성하며 [12-13]은 ShEx 스키마를 생성한다. [10-13]은 모두 기계를 통한 그래프 구조 검증이 가능한 수준의 스키마 생성을 위해 Direct Mapping의 매

핑 규칙과 매핑 대상 RDB의 스키마 정보를 함께 사용했다는 공통점이 있다.

[14-16]은 R2RML 방식으로 구축된 RDF 그래프에 대한 스키마를 자동 생성하기 위한 연구들이다. [14]에서는 그래프 스키마 생성을 위한 알고리즘만을 제안했으나 [15-16]은 그래프 스키마를 생성하는 시스템까지 구현한 사례들이다. [15]는 SHACL 스키마를 생성하며 [16]은 ShEx 스키마를 생성한다. [15-16]은 모두 R2RML로 작성된 매핑 규칙과 RDB 스키마를 함께 사용하여 기계를 통한 그래프의 구조 검증이 가능한 수준의 스키마를 생성한다.

RML을 사용하여 구축된 RDF 그래프를 위한 스키마를 자동 생성하기 위한 연구는 RML 사양을 개발한 조직에서 최근 발표한 [17]이 대표적이다. 이 연구에서는 SHACL 스키마를 생성해 주는 시스템을 제안했다. 이 연구는 그래프 스키마를 RML 매핑 규칙에서 수집한 정보만을 이용하여 생성하며 이 때문에 그 스키마는 기계를 통한 그래프 구조 검증 용도로 사용하기에는 제약 조건 구성이 충분치 않다는 한계가 있다.

2. Requirements

본 절에서는 제안 시스템이 기계를 통한 그래프 구조 검증이 가능한 수준의 스키마를 제공하기 위해 갖추어야 할 네 가지 요구사항을 제시하고자 한다. 요구사항들은 [17]에서 발견된 문제점에 대한 분석을 바탕으로 도출되었다.

첫째, 스키마를 구성하는 각각의 형상에 cardinality 제약 조건을 추가해야 한다. cardinality는 특정 형상이 그래프에서 출현할 수 있는 회수의 범위에 대한 제약 조건이다. 이 제약 조건은 매핑에 사용되는 데이터의 양태에 대한 분석을 통해서만 획득될 수 있다. [17]은 이러한 입력 데이터에 대한 양태 분석을 수행하지 않으므로 cardinality가 생략된 스키마를 생성한다. SHACL 검증기는 이 경우의 cardianlity를 '0 이상'으로 해석하기 때문에, 검증은 항상 성공하게 된다는 문제가 발생한다.

둘째, Open-world assumption(OWA) 방식뿐만 아니라 Closed-world assumption(CWA) 방식의 검증도 지원하는 스키마를 생성할 수 있어야 한다. [17]은 OWA 방식의 검증을 발동시키는 스키마만을 생성한다. OWA 방식의 검증은 형상에 명시되지 않은 구조 외에 추가적인 다른 구조가 덧붙여져 있는 것을 허용하는 반면 CWA 방식의 검증은 이를 허용하지 않는다. SHACL과 ShEx 둘 다 두 방식의 검증을 모두 지원하므로 사용자가 목적에 따라 선택할 수 있도록 두 방식의 검증을 위한 스키마를 제공할 수 있어야 한다.

셋째, SHACL 스키마 생성 시 repeated properties 구조의 형상에는 qualified value shape 제약 조건이 추가되어야 한다. repeated properties 구조의 형상이란 한 정점에 서로 다른 제약 조건으로 묘사되는 정점들이 동일 이름의 간선으로 연결되어 있는 구조의 그래프 검증에 대비하기 위한 형상이다. 즉, 한 형상 내에 동일한 간선 제약 조건이 여러 번 출현하는 구조를 일컫는다. 연구 [17]은 이러한 구조의 형상에 qualified value shape 제약 조건을 사용하지 않았다. 이 제약 조건이 없으면 SHACL 검증기는 간선 제약 조건을 만족하는 간선에 따라오는 모든 정점을 검증 대상으로 삼는다. 즉, 서로 다른 정점 제약 조건으로 나누어 검증돼야 하는 정점들이 단지 간선의 이름이 같은 이유로 모두 하나의 검증 대상으로 묶였기 때문에 검증은 늘 실패하게 된다. 이러한 문제를 예방하기 위해서는 qualified value shape 제약 조건을 추가하여 검증 대상을 구분해 주어야 한다. ShEx는 SHACL과 달리 repeated properties 구조에 대비한 별도의 추가적인 구문 구사를 요구하지 않는다.

넷째, 서로 다른 triples map에 의해 생성된 부분 그래프들이 합쳐진 구조를 형성했을 때를 대비한 형상도 제공되어야 한다. RML 매핑 규칙은 triples map을 단위로 조직된 것이며 RML 매핑은 triples map을 단위로 그래프를 생성한다. 같은 triples map에 의해 생성된 부분 그래프들은 모두 같은 구조를 갖는다. [17]은 이러한 특성에 근거하여 각각의 triples map으로부터 이에 대응하는 하나씩의 형상을 생성한다. 그러나 서로 다른 triples map에 의해 생성된 부분 그래프에 동일한 정점이 함께 존재하면 그 정점에 의해 그 그래프들은 하나로 병합된다. [17]은 이러한 병합된 구조를 검증할 형상을 제공하지 않는다.

III. The Proposed System

1. System Outline

그림 1은 제안 시스템의 구성 요소와 동작 과정을 요약한다. 4단계로 구분하여 동작 과정을 기술하고자 한다. 1단계는 RMLModelBuilder가 RML 문서로부터 RML 모델을 구축한다. RML 모델은 RML 문서를 메모리로 옮긴 것이다. RML 모델은 RML 사양에서 정의한 RML 어휘들의 논리 구조를 모사한 객체 모델로서 2~3단계에서 RML 사양에서 정의한 매핑 규칙의 논리 구조대로 이 모델을 탐색해 가며 RML 문서의 내용을 획득할 수 있게 한다.

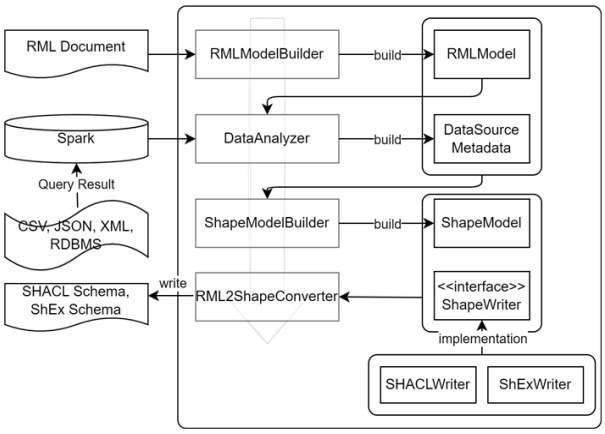

Fig. 1. System Outline

2단계는 DataAnalyzer가 매핑 대상 데이터의 양태를 분석하여 그 결과를 DataSourceMetadata에 기록한다. 분석 대상은 DataAnalyzer가 RML 모델에서 수집한다. 분석 결과는 그래프 형상을 구성하는 제약 조건들의 값으로 사용된다. DataAnalyzer는 매핑 대상 데이터를 오픈소스 분산 데이터 분석 엔진인 Apache Spark에 로딩한 후 분석을 수행한다. Spark를 사용하게 된 이유는 데이터 규모에 따른 시스템의 scalability를 Spark를 통해 보장받기 위함이며 RML이 지원하는 다양한 입력 데이터의 포맷을 Spark 또한 모두 지원하기 때문에 상이한 포맷의 데이터들을 Spark 상에서 일관된 방식으로 분석할 수 있다는 장점을 취하기 위함이었다.

3단계는 ShapeModelBuilder가 1~2단계의 결과인 RML 모델과 DataSourceMetadata를 입력으로 취하여 공통 형상 모델인 ShapeModel을 객체 모델 형태로 구축한다. ShapeModel은 특정 그래프 스키마 언어의 신택스에 종속적인 구조는 아니지만 시스템의 최종 출력 언어로 SHACL 혹은 ShEx 중 어느 언어가 선택되더라도 제약 조건의 값으로 사용될 데이터를 충분히 보유하고 있다. 이 공통 모델은 ShapeWriter라고 명명된 하나의 인터페이스를 제공한다. 이 인터페이스에는 3개의 추상 메소드가 정의되어 있다. 그림 1의 SHACLWriter와 ShExWriter는 이 인터페이스를 구현한 클래스들이다. 즉, 각각의 언어별로 이 3개의 메소드를 구현하는 것만으로 공통 형상 모델의 데이터를 해당 언어의 신택스에 맞게 나열하는 방법을 모두 지정한 효과를 낳는다.

4단계는 RML2ShapeConverter가 공통 형상 모델에서 출력에 사용될 요소들을 ShapeWriter의 메소드에 매개변수로 각각 전달한 뒤 반환받은 문자열들을 순서대로 파일에 씀으로써 스키마 파일을 최종적으로 생성하게 된다.

2. RML Model

본 절에서는 그림 2~3을 바탕으로 RML 모델의 구조를 기술하고자 한다. RML 모델의 모든 요소들의 명칭과 관계는 RML 사양과 동일하다.

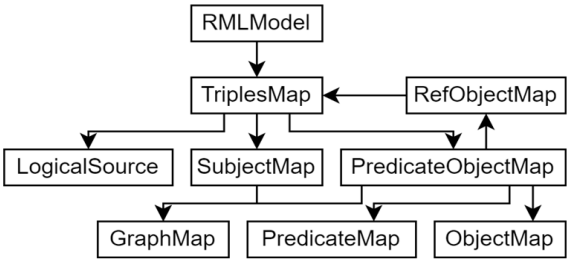

Fig. 2. RML Model's Core Classes

RML 모델은 triples map의 집합이다. triples map 하나가 동일 구조의 트리플들을 생성하기 위한 매핑 규칙이다. 트리플은 RDF에서 주어·술어·목적어로 구성된 문장을 의미한다. 주어·술어·목적어는 각각 그래프에서의 정점·간선·정점에 대응한다. triples map에는 해당 트리플 생성 규칙을 적용할 데이터셋 지정을 위한 1개의 logical source가 존재한다. triples map은 주어 생성 규칙인 1개의 subject map과 0개 이상의 술어·목적어 생성 규칙인 predicate object map으로 구성된다. predicate object map은 1개 이상의 술어 생성 규칙인 predicate map과 1개 이상의 목적어 생성 규칙인 object map 혹은 referencing object map으로 구성된다. object map은 새로운 목적어를 소스 데이터로부터 생산하고자 할 때 사용되며 referencing object map은 다른 triples map으로 생성된 트리플의 주어를 현재 생성 중인 트리플의 목적어로 취하고자 할 때 사용된다. 이러한 이유로 그림 2의 RefObjectMap이 TriplesMap을 참조하고 있다. 단일 predicate object map에 n개의 술어 생성 규칙과 m개의 목적어 생성 규칙이 정의되어 있다면 $n \times m$개의 술어·목적어 생성 규칙을 정의한 것이 된다. 마지막으로 subject map과 predicate object map에는 각각 0개 이상의 graph map이 정의될 수 있다. graph map이 존재하면 트리플이 아닌 주어·술어·목적어·그래프명으로 구성된 쿼드[18]라는 문장을 생성한다. 쿼드는 첫 세 성분인 트리플이 마지막 성분인 그래프명에 해당하는 그래프의 소속 트리플임을 밝히는 것이다. graph map은 그래프명 생성 규칙이다.

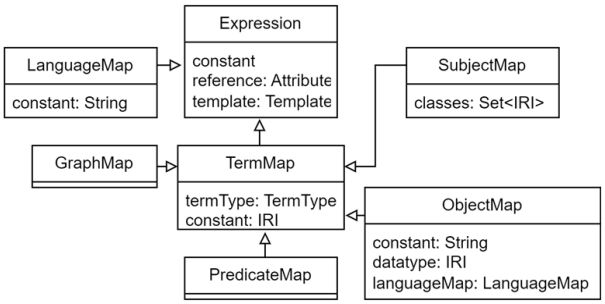

Fig. 3. Class Expression and its subclasses

그림 3은 RDF 문장 성분 생성 규칙에 해당하는 클래스들의 상속 관계를 표현한다. Expression은 이들의 최상위 클래스다. 최상위 클래스의 세 변수는 RDF 문장 성분 생성 방식에 대응한다. RML에서는 모든 RDF 문장 성분을 세 가지 방식 중 하나로 생성한다. reference에는 데이터셋의 한 속성명이 할당되며 이 경우 할당된 속성에 해당하는 데이터가 그대로 문장 성분으로 출력된다. template에는 포맷 문자열이 할당된다. 포맷 문자열에는 데이터셋의 속성명이 여럿 포함되어 있을 수 있다. 이 경우 속성명을 속성값으로 모두 치환시킨 문자열들이 문장 성분으로 출력된다. constant에는 RML 문서에 명시한 고정된 값이 할당된다. 이 경우 constant에 할당된 값이 그대로 문장 성분으로 출력된다.

주어·술어·목적어·그래프명 생성 규칙에 해당하는 클래스들의 공통 상위 클래스는 TermMap이다. TermMap에 있는 termType은 생성될 문장 성분의 타입을 값으로 한다. IRI(Internationalized Resource Identifier), 블랭크 노드, 리터럴이 termType의 후보다. RDF에서 주어의 타입은 IRI 혹은 블랭크 노드일 수 있다. 목적어는 세 타입 모두 가능하다. 술어와 그래프명은 IRI만 가능하다. IRI는 모든 문장 성분에서 가능하므로 IRI 타입의 constant가 TermMap에 정의되어 있다. 블랭크 노드는 IRI의 일종이기에 블랭크 노드 타입의 constant는 IRI 타입의 constant로 처리된다. 리터럴은 목적어에서만 가능하므로 ObjectMap에 리터럴 타입의 constant가 정의되어 있다.

SubjectMap의 classes는 IRI 집합이다. 이 집합이 공집합이 아니면 이 집합 크기만큼의 트리플이 생성된다. 이 트리플의 술어는 'rdf:type'으로 미리 정해져 있으며 주어는 SubjectMap에 의해 생성된 주어이고 목적어는 classes의 원소 IRI들이다. 이 트리플은 SubjectMap에 의해 생성된 주어가 RDF에서의 무슨 클래스에 속하는지를 뜻한다. 이처럼 주어 생성 규칙만으로도 트리플 생성이 가능하므로 술어·목적어 생성 규칙이 없는 triples map도 가능하다.

RDF 리터럴의 구성 요소는 값, 데이터 타입 IRI, 언어 태그이다. 값만으로도 사용될 수 있으며 값에 다른 요소도 덧붙여 사용될 수도 있다. 예를 들어, "Bob"은 값만 사용한 사례이며, "Bob"^^xsd:string은 데이터 타입 IRI를 덧붙인 사례이며, "Bob"@en은 언어 태그를 덧붙인 사례이다. ObjectMap에는 리터럴의 부가 요소를 저장할 datatype, languageMap 변수가 정의되어 있다. languageMap 변수의 타입인 LanguageMap은 Expression의 하위 클래스이기에 언어 태그 또한 Expression이 지원하는 세 가지 방식으로 생성 가능하다.

3. Analysis of Input Datasets

본 절에서는 본격적인 데이터 분석을 시작하기에 앞서 수행하는 평탄화 작업을 먼저 소개한 뒤 데이터 분석을 통해 얻어내는 사항들을 정리하여 제시하고자 한다. 본 절에서 기술되는 평탄화 작업과 데이터 분석 작업은 Spark 상에서 Spark가 제공하는 API를 사용하여 작성된 코드의 실행으로 수행된다.

(a)
{"Name": "John",
"Address": {"Street": "123 Main St", "City": "Anytown"},
"Phones": ["123-456-7890", "987-654-3210"]}

(b)

Name	Address	Phones
John	{"Street": "123 Main St", "City": "Anytown"}	["123-456-7890", "987-654-3210"]

(c)

Name	Address.Street	Address.City	Phones
John	123 Main St	Anytown	123-456-7890
John	123 Main St	Anytown	987-654-3210

Fig. 4. Flattening a JSON Object

RML이 지원하는 데이터 포맷은 CSV, JSON, XML, RDB다. 이 중에서 CSV와 RDB는 테이블 구조이고 JSON과 XML은 트리 구조이다. RML 매핑에서 테이블 구조 데이터의 경우 행 단위로 매핑을 수행한다. 즉 하나의 행 데이터를 대상으로 1회의 triples map 적용이 발생한다. 반면 RML 매핑에서 트리 구조 데이터에 대해서는 객체 단위로 매핑을 수행한다. 객체는 속성으로서 리터럴, 객체, 배열이 가능하며 배열의 요소 또한 리터럴, 객체, 배열이 가능하다. 이러한 객체 구조의 복잡성으로 인해 한 객체에 대한 트리플 생성 규칙 적용이 여러 번 발생할 수 있다. 제안 시스템은 cardinality 제약 조건 생성을 위해 객체에 대한 정확한 트리플 생성 규칙 적용 횟수를 알아 내야 한다. 이를 위해 제안 시스템은 트리 구조를 테이블 구조로 변형한다. 이 작업이 평탄화 작업이다.

그림 4는 JSON 데이터를 대상으로 한 평탄화 작업 과정을 요약한다. JSON과 XML은 구문의 차이만 존재하는 DOM(Document Object Model)이라는 객체를 텍스트로 표현하기 위한 모델의 서로 다른 구현일 뿐이라서 JSON 포맷을 기준으로 평탄화 작업을 기술한다. 그림 4의 (a)는 매핑 단위로 선정된 JSON 객체 예다. 실제 JSON 데이터셋은 (a) 객체와 구조가 동일한 객체들의 집합이다. (a)의 Name 속성은 값이 리터럴인 예이고, Address 속성은 값이 객체인 예이며, Phones 속성은 값이 배열인 사례로서 요소가 리터럴이다. 그림 4의 (b)는 (a) 객체가 Spark에 로딩된 모습이다. Spark는 데이터를 테이블 구조로 관리하며 객체의 요소값 종류와 상관없이 하나의 셀 값으로 저장한다. 그림 4의 (c)는 (b)로부터 평탄화 작업을 완료한 모습이다. 평탄화 작업은 셀 값이 객체일 땐 객체 속성 각각을 컬럼으로 분해하며, 셀 값이 배열일 땐 요소 각각을 행으로 분해한다. 이 작업을 위한 코드는 재귀적인 논리로 작성되었기 때문에 다양한 깊이의 트리 구조 객체를 처리할 수 있다. 평탄화된 테이블의 컬럼명들은 RML 매핑 문서에서 참조될 수 있는 객체의 속성명 후보를 모두 나열한 것과 일치한다. 또한 평탄화된 테이블의 각각의 행은 실제 매핑 규칙이 1회 적용되는 대상 데이터와 일치한다.

Table 1. Datasets Analysis Results

Metadata	Where to use
data type	literal object
length range	subject/object regex, string literal object
value range	numeric literal object
cardinality range	predicate-object pair

표 1은 제안 시스템이 데이터 분석을 통해 추출 해내는 네 가지 메타데이터와 메타데이터 각각의 용처를 요약한다. 첫 번째 메타데이터는 데이터 타입이다. 데이터 타입은 reference 방식으로 리터럴 목적어를 생성하는 컬럼들을 대상으로 Spark 상에서 추론을 통해 결정한다. 이 메타데이터는 그래프 스키마에서 리터럴 목적어의 타입 제한을 위한 제약 조건으로 사용된다. 두 번째 메타데이터는 문자열 길이 범위다. 문자열 길이 범위는 reference 방식으로 문자열 리터럴 목적어를 생성하는 컬럼들과 template 방식으로 주어 혹은 목적어를 생성하는 컬럼들 각각을 대상으로 추출 해낸다. 이 메타데이터는 그래프 스키마에서 문자열 리터럴의 길이에 대한 최소·최대 제약 조건으로 사용된다. 또한 template 방식으로 생성되는 주어 혹은 목적어에 대한 제약 조건으로는 정규식(regular expression)이 사용되는데 이 정규식은 template 내 포맷 문자열을 기반으로 만들어진 것이다. 이때 포맷 문자열 내 컬럼명은 해당 컬럼으로부터 추출한 길이 정보로 대치된다. 세 번째 메타데이터는 최소·최대 숫자 값이다. 수 범위는 reference 방식으로 숫자 리터럴 목적어를 생성하는 컬럼들을 대상으로 얻어낸다. 이 메타데이터는 그래프 스키마에서 숫자 리터럴 목적어의 값 범위 제약 조건으로 사용된다. 마지막 메타데이터는 cardinality다. 이 메타데이터는 주어·술어·목적어 생성 규칙 조합 각각에 대하여 구한다. 고유의 구조마다 생성되는 트리플 개수의 범위를 제약 조건으로 명시하기 위함이다. cardinality를 결정하기 위한 연산을 요약하면 다음과 같다. 각 조합의 주어·술어·목적어 생성에 사용되는 컬럼들로만 구성된 테이블을 우선 만들고 null이 포함된 행은 제거한다. null이 있는 행은 트리플 생성에서 제외되기 때문이다. 또한 행의 중복도 제거한다. RDF에서는 중복된 트리플을 허용하지 않기 때문이다. 그다음 주어를 생성하는 컬럼값을 기준으로 행들을 그룹 짓고 그룹별 행수를 구한 뒤 행수의 최소·최댓값을 cardinality로 취한다. referencing object map으로 목적어가 생성되는 경우 목적어가 될 주어를 생성하는 다른 데이터셋과의 조인 연산이 가장 먼저 수행된다.

4. Common Shape Model

본 절에서는 형상 모델의 구조와 형상 모델을 구축하는 절차를 기술하고자 한다.

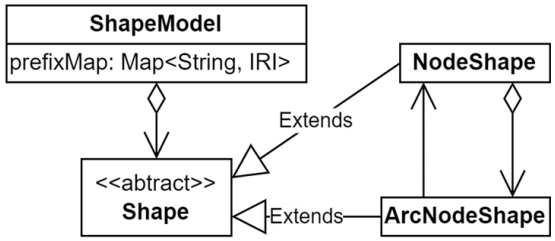

Fig. 5. The Structure of Common Shape Model

그림 5는 형상 모델의 구조를 보여주는 클래스 다이어그램이다. 형상 모델인 ShapeModel은 Shape 즉 형상의 집합이다. Shape은 추상 클래스기 때문에 구체적인 형상은 Shape의 하위 타입인 NodeShape 혹은 ArcNodeShape의 객체들이다. 우선 NodeShape은 주어 자격 정점의 형상을 제약할 수 있는 세부 조건들로 구성되어 있다. 주어를 묘사하는 NodeShape 객체는 그 주어에 연결될 서로 구분되는 술어·목적어 구조를 묘사하는 ArcNodeShape 객체들을 참조하는 구조다. ArcNodeShape은 술어 제약 조건과 목적어 제약

조건으로 구성된다. 목적어 또한 정점이므로 목적어 제약 조건도 NodeShape 객체다. 즉 ArcNodeShape은 자신의 목적어를 묘사하는 NodeShape 객체를 참조하는 구조다. 이러한 이유로 그림 5에서 NodeShape과 ArcNodeShape 두 클래스가 상호 참조 관계로 표현된 것이다.

1. RefObjectMap에 의해 참조되는 TriplesMap 목록 생성
 → 참조되는 TriplesMap 당 '대표 id' 생성
2. 같은 주어를 생성할 가능성이 있는 TriplesMap끼리 클러스터링
3. 각 클러스터별로 클러스터 멤버들의 모든 조합 구하기
4. 모든 클러스터의 각각의 조합에 대응하여 Shape 생성
 → SubjectMap으로부터 NodeShape생성
 → SubjectMap의 classes 속성, PredicateObjectMap } 으로부터 ArcNodeShape 생성
5. 참조되는 TriplesMap이 속한 조합에서 생성된 NodeShape들을 OR 연산자로 연결한 NodeShape 생성
 → '대표 id'를 이 NodeShape의 id로 할당

Fig. 6. Process to Build a Common Shape Model

그림 6은 공통 형상 모델을 구축하는 절차를 요약한다. 절차 1은 RML 모델에서 RefObjectMap의 사용으로 인해 다른 TriplesMap에 의해 참조되는 TriplesMap들을 찾아 목록을 생성한 뒤 이 목록의 TriplesMap 당 1개씩의 '대표 id'라고 명명한 향후 생성될 NodeShape의 식별자를 미리 생성 해둔다. 형상의 생성보다 형상의 참조가 먼저 이루어지기 때문에 식별자부터 먼저 생성한 것이다.

절차 2는 RML 모델의 모든 TriplesMap을 대상으로 같은 주어를 생성할 가능성이 있는 TriplesMap끼리 클러스터를 구성한다. 동일 주어 생성 여부 판단은 SubjectMap의 네 속성 termType, constant, reference, template을 가지고 수행된다. 기본적으로 termType이 서로 일치해야 나머지 세 속성의 값을 따진다. 우선 template끼리는 포맷 문자열의 패턴이 일치할 때, constant끼리는 값이 일치할 때, reference 컬럼끼리는 문자열 길이 범위 간 겹치는 구간이 있을 때 동일 주어가 생성될 수 있다고 판정한다. 그다음 template 포맷 문자열 패턴에 constant 값이 매칭될 때와 template 포맷 문자열 패턴의 최소 길이가 reference 컬럼의 문자열 길이 범위에 있을 때 동일 주어가 생성될 수 있다고 판정한다. 마지막으로 constant 값의 문자열 길이가 reference 컬럼의 문자열 길이 범위에 있을 때도 동일 주어가 생성될 수 있다고 판정한다.

절차 3은 절차 2에서 생성한 각각의 클러스터를 대상으로 클러스터 멤버의 모든 조합을 구한다. 예를 들어, RML 모델에 네 개의 TriplesMap T_1~T_4가 있고 이들을 가지고 절차 2에서 두 개의 클러스터 $\{T_1,\ T_2,\ T_3\}$, $\{T_4\}$가 생성되었다면, 절차 3에서는 클러스터 $\{T_1,\ T_2,\ T_3\}$으로부터 7개의 조합 $\{T_1\}$, $\{T_2\}$, $\{T_3\}$, $\{T_1,\ T_2\}$, $\{T_1,\ T_3\}$, $\{T_2,\ T_3\}$, $\{T_1,\ T_2,\ T_3\}$을 생성하고 클러스터 $\{T_4\}$으로부터 1개의 $\{T_4\}$을 생성한다. 즉 각 클러스터를 집합으로 상정하고 공집합을 제외한 모든 부분 집합을 구한 것이다.

절차 4는 절차 3에서 생성한 조합 각각으로부터 형상을 생성한다. 이 형상은 주어 구조를 위한 NodeShape과 이에 연결될 술어·목적어 구조별 ArcNodeShape으로 구성된다. NodeShape은 조합의 SubjectMap으로부터 생성되며 조합 멤버가 복수일 경우 동일 주어 생성 가능성을 가늠할 때 도출된 서로 겹치는 속성을 사용해 생성된다. ArcNodeShape은 조합의 모든 PredicateObjectMap과 SubjectMap의 classes 속성으로부터 생성된다. classes 속성으로도 술어·목적어가 생성되기 때문이다. 절차 4에서는 repeated properties 구조에 대비하기 위해 같은 NodeShape에 연결될 ArcNodeShape들을 대상으로 술어 제약 조건이 같은 것들이 존재하는지를 점검한 뒤 이러한 구조가 발견되면 해당하는 ArcNodeShape마다 isRepeatedProperty라는 속성의 값을 true로 세팅 해둔다. 이 값에 따라 SHACL로 표현 시 서로 다른 SHACL 어휘가 선택될 수 있도록 하기 위함이다.

절차 4에서는 생성하는 모든 형상에 식별자를 부여한다. SHACL과 ShEx 모두 식별자를 통한 형상 참조를 허용하기 때문이다. NodeShape의 참조는 RefObjectMap에 의해 발생한다. 이 경우 RefObjectMap에 의해 참조되는 TriplesMap의 SubjectMap으로부터 생성된 NodeShape이 목적어에 대한 형상이므로 목적어 형상 위치에 이 NodeShape의 식별자가 배치되면 된다. 이 용도의 식별자가 절차 1에서 생성한 '대표 id'다. 대표 id를 부여받는 NodeShape은 해당 NodeShape을 생성케 한 TriplesMap이 다른 TriplesMap과 같은 주어를 생성할 가능성에 따라 달라진다. 앞선 예에서 T_4처럼 같은 주어를 생성할 가능성이 없는 TriplesMap은 절차 4에서 조합 $\{T_4\}$로부터 생성되는 NodeShape의 식별자로 부여된다. 반면 앞선 예에서 T_1처럼 같은 주어를 생성할 가능성이 있는 TriplesMap은 절차 5에서 생성되는 NodeShape의 식별자로 부여된다. 이 NodeShape은 절차 4에서 T_1이 속한 조합 $\{T_1\}$, $\{T_1,\ T_2\}$, $\{T_1,\ T_3\}$, $\{T_1,\ T_2,\ T_3\}$ 각각에서 생성한 네 개의 NodeShape의 식별자를 OR 연산자로 연결한 것이다. 이것은 목적어의 형상이 이 네 가지 중 하나기 때문이다.

5. Serialization of Common Shape Model

```
public interface ShapeWriter {
    String writeDirectives(Map<String, IRI> prefixMap);
    String writeNodeShape(NodeShape nodeShape);
    String writeArcNodeShape(ArcNodeShape arcNodeShape);
}
```

Fig. 7. Interface for Shape Serialization

그림 7은 그래프 스키마 언어들이 공통 형상 모델을 자신의 신택스로 표현하고자 할 때 구현해야 하는 세 개의 메소드를 정의하고 있는 인터페이스다. 그래프 스키마 문서에는 필연적으로 수많은 IRI가 출현하므로 가독성을 위해 축약된 형태의 IRI를 사용한다. 축약된 형태의 IRI를 사용하기 위해서는 문서에서 특정 IRI를 대신할 접두어를 미리 선언해야 한다. 각각의 언어는 writeDirectives에서 자신의 신택스대로 이 선언부에 해당하는 문자열을 반환하도록 구현하면 된다. 이 메소드에 매개변수로서 전달되는 선언부 구성에 필요한 접두어와 IRI 쌍들은 그림 5에 표현된 ShapeModel의 속성들이다. writeNodeShape과 writeArcNodeShape 각각에는 형상 모델의 NodeShape과 ArcNodeShape이 하나 전달되었을 때 그것을 자신의 신택스로 출력하기 위한 코드를 작성해 두면 된다.

그래프 스키마 출력은 그림 1의 RML2ShapeConverter가 담당한다. 이 컨버터는 우선 writeDirective 호출을 통해 이름공간 선언부를 획득하여 파일에 쓴다. 그다음 형상 모델에 등록된 주어를 위한 NodeShape 각각을 가지고 writeNodeShape을 호출하여 획득한 문자열을 파일에 순서대로 씀으로써 그래프 스키마 문서 생성을 종료한다. 주어를 위한 NodeShape에는 자신의 술어·목적어를 위한 ArcNodeShape들의 참조를 가지고 있으므로 술어·목적어를 위한 구문이 필요한 위치에서 NodeShape은 writeArcNodeShape을 호출하여 해당 구문을 획득하여 사용하는 방식이며 연쇄적으로 ArcNodeShape 또한 자신의 목적어를 위한 NodeShape의 참조를 가지고 있으므로 목적어를 위한 구문이 필요한 위치에서 ArcNodeShape은 writeNodeShape을 호출하여 해당 구문을 획득하여 사용하는 방식으로 동작한다.

Table 2. RML-to-Shape Terms Mapping

RML	SHACL	ShEx	S	P	O
termType	nodeKind		✓		✓
classes constant	path	predicate		✓	
classes	hasValue	objectValue			✓
template	pattern		✓		✓
constant	hasValue	objectValue	✓		✓
datatype	datatype				✓
languageMap	languageIn	Language			✓
parentTriplesMap	node	tripleExprRef			✓
reference	Refer to Table 1.				

표 2는 RML 모델의 속성이 SHACL과 ShEx 두 그래프 스키마 언어의 무슨 제약 조건 생성에 반영되었는지를 보여주며 또한 각각의 제약 조건이 트리플의 무슨 문장 성분을 제약하는 것인지도 표시했다. 표 2의 마지막 행은 reference 속성이 표 1의 메타데이터 카테고리의 의미에 해당하는 제약 조건 생성에 사용되었음을 뜻한다. 표 2에서 SHACL 열의 값들은 SHACL 어휘이며 ShEx 열의 값들은 ShEx 문장 성분명이다. SHACL은 SHACL 스키마가 RDF 그래프가 되도록 설계하였다. 즉 SHACL 스키마는 트리플로 구성된다. 형상의 식별자가 주어, 형상을 구성하는 제약 조건명이 술어, 조건의 값이 목적어가 된다. 표 2에서의 SHACL 어휘란 술어 자격의 제약 조건명들이다. 반면 ShEx는 신택스가 BNF(Backus-Naur Form)으로 정의된 언어라서 문맥에 따라 값이 출현한 위치로 값이 뜻하는 제약 조건 종류가 표현된다.

6. Demonstration of Graph Schema Creation

```
1: ID, Sport, Name
2: 10, 100, Venus Williams
3: 20, ,Demi Moore
      [student.csv]

1: ID, Name
2: 100, Tennis
      [sport_en.csv]

1: ID, Name
2: 100, Tenis
      [sport_es.csv]
```

Fig. 8. Input Files

본 절에서는 그림 8~12를 통해 제안 시스템이 2장에서 기술한 네 가지 요구사항을 모두 만족시키는 그래프 스키마를 생성했음을 보이는 사례를 기술하고자 한다. 그림 8은 매핑의 입력 데이터로 사용된 세 개의 CSV 파일의 내용이다. 각 파일의 1행은 컬럼명이다. 그림 9는 RML 문서다. 그림 10은 매핑 결과인 RDF 그래프다. 그림 11은 입

력 데이터와 RML 문서를 바탕으로 제안 시스템이 생성한 결과 그래프의 구조를 검증할 수 있는 ShEx 스키마이고 그림 12는 동일 내용의 SHACL 스키마다.

```
01: @prefix rr: <http://www.w3.org/ns/r2rml#> .
02: @prefix foaf: <http://xmlns.com/foaf/0.1/> .
03: @prefix rml: <http://semweb.mmlab.be/ns/rml#> .
04: @prefix ql: <http://semweb.mmlab.be/ns/ql#> .
05: @prefix ex: <http://ex.com/> .
06: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
07: @base <http://ex.com/base/> .

08: <TM1> a rr:TriplesMap;

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

11:   rr:subjectMap [ rr:template "http://ex.com/student{ID}" ];

12:   rr:predicateObjectMap [
13:     rr:predicate foaf:name ;
14:     rr:objectMap [ rml:reference "Name"; ] ];

15:   rr:predicateObjectMap [
16:     rr:predicate ex:practises ;
17:     rr:objectMap [
18:       a rr:RefObjectMap ;
19:       rr:parentTriplesMap <TM2>;
20:       rr:joinCondition [ rr:child "Sport" ; rr:parent "ID" ; ]]].

21: <TM2> a rr:TriplesMap;
22:   rml:logicalSource [
23:     rml:source "sport_en.csv"; rml:referenceFormulation ql:CSV ];

24:   rr:subjectMap [ rr:template "http://ex.com/sport{ID}" ];

25:   rr:predicateObjectMap [
26:     rr:predicate rdfs:label ;
27:     rr:objectMap [ rml:reference "Name"; rr:language "en" ]; ].

28: <TM3> a rr:TriplesMap;
29:   rml:logicalSource [
30:     rml:source "sport_es.csv"; rml:referenceFormulation ql:CSV ];

31:   rr:subjectMap [ rr:template "http://ex.com/sport{ID}" ];

32:   rr:predicateObjectMap [
33:     rr:predicate rdfs:label ;
34:     rr:objectMap [ rml:reference "Name"; rr:language "es" ]; ].
```

Fig. 9. RML Document

그림 9는 TM1, TM2, TM3로 명명된 세 개의 triples map으로 구성되어 있다. 8~20행이 TM1이고, 21~27행이 TM2이며, 28~34행이 TM3이다. TM1은 strudent.csv를 가지고, TM2는 sport_en.csv를 가지고, TM3는 sport_es.csv를 가지고 트리플을 생성한다.

```
1: @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3: @prefix ex: <http://ex.com/> .

4: ex:student10 foaf:name "Venus Williams" .
5: ex:student10 ex:practises ex:sport100 .
6: ex:student20 foaf:name "Demi Moore" .
7: ex:sport100 rdfs:label "Tennis"@en .
8: ex:sport100 rdfs:label "Tenis"@es .
```

Fig. 10. Result RDF Graph

그림 10의 4~6행 트리플은 TM1이 생성한 것이다. 그중 4~5행 트리플은 student.csv의 2행 데이터로 생성된 것이고 6행 트리플은 3행 데이터로 생성된 것이다. 3행 데이터는 sport 열 데이터가 없는 관계로 ex:practises가 술어인 트리플을 생성하지 않았다. 그림 10의 7행 트리플은 TM2가 sport_en.csv의 2행 데이터로 그리고 8행 트리플은 TM3가 sport_es.csv의 2행 데이터로 생성한 것이다. 7행과 8행 트리플은 서로 다른 triples map이 생성했으나 같은 주어를 생성한 사례다. 5행 트리플의 목적어인 ex:sport100은 매핑 규칙에 따라 7행 트리플의 주어를 목적어로 취한 것이지만 7행과 8행 주어가 중복되어 8행 트리플의 주어도 동시에 목적어로 취한 구조가 형성되었다.

```
01: PREFIX my: <http://my.ex/ns#>
02: PREFIX ex: <http://ex.com/>
03: PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
04: PREFIX foaf: <http://xmlns.com/foaf/0.1/>

05: my:S1 IRI /^http:\/\/ex\.com\/student(.{2,})$/ AND CLOSED {
06:   $my:P1 foaf:name LITERAL MINLENGTH 10 MAXLENGTH 14
07:   $my:P2 ex:practises @my:SO1 ?;
08: }

09: my:S2 IRI /^http:\/\/ex\.com\/sport(.{3,})$/ AND CLOSED {
10:   $my:P3 rdfs:label [@en] LENGTH 6
11: }

12: my:S3 IRI /^http:\/\/ex\.com\/sport(.{3,})$/ AND CLOSED {
13:   $my:P4 rdfs:label [@es] LENGTH 5
14: }

15: my:SA1 IRI /^http:\/\/ex\.com\/sport(.*)$/ AND CLOSED {
16:   $my:P5 rdfs:label [@en] LENGTH 6;
17:   $my:P6 rdfs:label [@es] LENGTH 5
18: }

19: my:SO1 @my:S2 OR @my:SA1
```

Fig. 11. ShEx Schema for Fig. 10

그림 11~12에서 'my'가 접두어인 S1~S2, P1~P6, SA1, SO1은 모두 형상의 식별자 IRI다. S1~S2, SA1, SO1은 주어 구조를 제약하는 NodeShape의 식별자이고 P1~P6는 술어·목적어 구조를 제약하는 ArcNodeShape의 식별자이다. 두 그림에서 식별자가 서로 같은 것들은 공통 형상 모델의 동일한 형상을 구문만 달리하여 출력된 관계다.

```
01: @prefix my: <http://my.ex/ns#> .
02: @prefix ex: <http://ex.com/> .
03: @prefix sh: <http://www.w3.org/ns/shacl#> .
04: @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
05: @prefix foaf: <http://xmlns.com/foaf/0.1/> .

06: my:S1 a sh:NodeShape ; sh:closed true ;
07:    sh:nodeKind sh:IRI ;
08:    sh:pattern "^http://ex.com/student(.{2,})$" ;
09:    sh:property my:P1 ;
10:    sh:property my:P2 .

11: my:P1 a sh:PropertyShape ;
12:    sh:path foaf:name ;
13:    sh:nodeKind sh:Literal ;
14:    sh:minLength 10 ; sh:maxLength 14 ;
15:    sh:minCount 1 ; sh:maxCount 1 .

16: my:P2 a sh:PropertyShape ;
17:    sh:path ex:practises ;
18:    sh:node my:SO1 ;
19:    sh:maxCount 1 .

20: my:S2 a sh:NodeShape ; sh:closed true ;
21:    sh:nodeKind sh:IRI ;
22:    sh:pattern "^http://ex.com/sport(.{3,})$" ;
23:    sh:property my:P3 .

24: my:P3 a sh:PropertyShape ;
25:    sh:path rdfs:label ;
26:    sh:languageIn ( "en" ) ;
27:    sh:minLength 6 ; sh:maxLength 6 ;
28:    sh:minCount 1 ; sh:maxCount 1 .

29: my:S3 a sh:NodeShape ; sh:closed true ;
30:    sh:nodeKind sh:IRI ;
31:    sh:pattern "^http://ex.com/sport(.{3,})$" ;
32:    sh:property my:P4 .

33: my:P4 a sh:PropertyShape ;
34:    sh:path rdfs:label ;
35:    sh:languageIn ( "es" ) ;
36:    sh:minLength 5 ; sh:maxLength 5 ;
37:    sh:minCount 1 ; sh:maxCount 1 .

38: my:SA1 a sh:NodeShape ; sh:closed true ;
39:    sh:nodeKind sh:IRI ;
40:    sh:pattern "^http://ex.com/sport(.*)$" ;
41:    sh:property my:P5 ;
42:    sh:property my:P6 .

43: my:P5 a sh:PropertyShape ;
44:    sh:path rdfs:label ;
45:    sh:qualifiedValueShape [
46:       sh:languageIn ( "en" ) ;
47:       sh:minLength 6 ; sh:maxLength 6 ] ;
48:    sh:qualifiedMinCount 1 ; sh:qualifiedMaxCount 1 .

49: my:P6 a sh:PropertyShape ;
50:    sh:path rdfs:label ;
51:    sh:qualifiedValueShape [
52:       sh:languageIn ( "es" ) ;
53:       sh:minLength 5 ; sh:maxLength 5 ] ;
54:    sh:qualifiedMinCount 1 ; sh:qualifiedMaxCount 1 .

55: my:SO1 a sh:NodeShape ; sh:or ( my:S2 my:SA1 ) .
```

Fig. 12. SHACL Schema for Fig. 10

S1~S3는 각각 TM1~TM3의 각각의 subject map에 대응하여 생성된 것이다. SA1은 동일 주어를 생성하는 조합 {TM2, TM3}의 subject map들로 생성된 것이다. SO1은 TM2가 TM1에서 참조되기 때문에 조합 {TM2}에서 생성된 S2와 조합 {TM2, TM3}에서 생성된 SA1을 OR 연산자로 연결한 형상이다. SO1은 그림 11의 7행과 그림 12의 18행에서 목적어 구조에 대한 형상으로서 참조되고 있다.

P1은 TM1 내 12~14행, P2는 TM1 내 15~20행, P3는 TM2 내 25~27행, P4는 TM3 내 32~24행의 predicate object map에 대응하여 생성된 것이다. P5와 P6는 repeated properties 구조에 대비하기 위하여 생성된 ArcNodeShape으로서 P5는 P3에 그리고 P6는 P4에 qualified value shape 구문을 적용한 것이다. 이 구문은 SHACL에서만 요구되는 구문이기에 ShEx에서는 P3와 P5 그리고 P4와 P6가 식별자만 다를 뿐 구문이 동일하다.

제안 시스템이 생성한 스키마와 [17]이 생성한 스키마를 가지고 그림 10 그래프를 검증한 결과를 비교하면 제안 시스템이 생성한 SHACL과 ShEx 스키마로는 그림 10의 4~6행 트리플을 S3가 검증해 냈으며 7~8행 트리플을 SA1이 검증해 냈다. 반면에 [17]이 생성한 스키마로 검증했을 땐 4~6행 트리플은 검증에 성공했으나 7~8행 트리플을 검증할 수 있는 형상은 존재하지 않았다. 7~8행 트리플을 검증하지 못한 이유는 [17]이 repeated properties 구조에 대한 대비가 없기 때문이다. 4~6행 트리플도 주어 구조만 제약 조건을 충족시켰음에도 불구하고 검증에 성공했다. 왜냐하면, [17]은 술어·목적어 구조에 대한 모든 cardinality가 '0 이상'이기 때문이다.

본 사례를 통해 제안 시스템이 생성한 스키마가 2장에서 기술한 네 가지 개선 사항을 다음과 같이 모두 충족시켰음을 확인할 수 있다. 첫째, P1~P6에서 보듯이 모든 ArcNodeShape에 cardinality가 출력되어 있음을 확인할 수 있다. 둘째, SA1과 S1~S3에서 보듯이 SHACL에서의 'sh:closed true'라는 구문과 ShEx에서의 'CLOSED'라는 예약어를 사용해 CWA 방식으로 검증하는 형상을 출력했음을 확인할 수 있다. 제안 시스템은 사용자 설정으로 CWA와 OWA를 선택할 수 있도록 하였다. 셋째, SA1에서 보듯이 repeated properties 구조에 대비한 형상을 생성했음을 확인할 수 있다. SA1이 없다면 그림 10의 7~8행 트리플은 S2와 S3만으로는 검증이 불가능하다. 넷째, SA1과 SO1을 통해 서로 다른 tripls map에서 동일 주어 생성으로 인해 트리플 구조가 합쳐지는 상황에 대한 대비한 형상의 생성과 이 상황에서 repeated properties 문제를 또다시 고려했음은 물론 합쳐진 구조가 참조되는 상황까지 대비한 형상도 생성했음을 확인할 수 있다.

IV. Conformance Test

본 장에서는 다양한 매핑 규칙 조합을 가지고 제안 시스템이 생성한 스키마의 정확성을 평가한다. 평가는 RML Test Cases[19]를 가지고 수행하였다. [19]는 RML 매핑 엔진이 정확한 RDF 그래프를 생성하는지를 평가하고자 개발된 테스트 케이스 세트다. [19]는 총 297개 테스트 케이스를 60개 범주로 구분해 놓은 구성이다. 각각의 테스트 케이스는 매핑의 입력인 데이터 파일, 매핑 규칙인 RML 문서, 매핑의 출력인 트리플이 담긴 RDF 파일로 구성되어 있다. 매핑 규칙 조합은 총 60가지로 각각의 범주에 대응한다. 한 범주에 속한 테스트 케이스들은 입력 파일의 포맷만 다르다. 제공되는 포맷이 CSV, JSON, XML, MySQL, PostgreSQL, SQLServer로 총 6종류이므로 한 범주에 최대 6개의 테스트 케이스가 속해있는 구성이다.

제안 시스템을 위한 평가는 테스트 케이스별로 다음과 같은 방법으로 수행했다. 우선 제안 시스템이 데이터 파일과 RML 문서를 가지고 스키마를 생성한다. 그다음 스키마와 RDF 파일을 검증기에 입력한다. 그다음 RDF 파일 내 모든 주어 노드 각각을 타겟 노드로 지정한다. 타겟 노드에 의해 검증 대상이 결정된다. 검증 대상은 주어인 타겟 노드를 포함하여 그 노드에 연결된 모든 술어·목적어들로 설정된다. 그다음 검증기가 CWA 방식의 검증을 수행한다. 이 방식에서는 검증 대상 구조와 정확히 일치하는 형상이 존재해야 그 대상이 검증을 통과할 수 있다. 이와 같은 방법을 SHACL 검증기와 ShEx 검증기로 각각 수행하였다. 본 평가에서는 모든 타겟 노드에 대한 검증이 성공해야 해당 테스트 케이스를 통과했다고 판정했다. 아래 표 3은 평가 결과를 요약한다. SHACL과 ShEx 간 검증 결과 차이는 없었다. 총 60개 범주 중 42개 범주는 검증에 성공했으며 나머지 18개 범주는 검증에 실패하였다. 실패 사유 또한 범주에 따라 아래 3가지 유형으로 정리되었다.

Table 3. Test Result Summary

Result Description	# of categories
validation success	42
1. schema creation failed	4
2. schema created but no graph	8
3. schema created but validation failed	6

유형 1 범주 테스트에서는 제안 시스템이 스키마 생성에 실패함으로써 검증조차 시도할 수 없었다. 이 유형을 유발한 4개 범주는 모두 RML 매핑 엔진이 그래프 생성에 실패해야 해당 테스트를 통과한 것으로 판정되는 테스트 케이스들로 구성되어 있다. 이 범주의 RML 문서에는 모두 각기 다른 구문 오류를 의도적으로 심어 놓았기 때문이다. 이 범주의 테스트에서 제안 시스템은 RML 모델 생성 도중 감지된 RML 구문 오류를 출력하고 처리를 종료했다.

유형 2 범주 테스트에서는 제안 시스템이 스키마 생성에는 성공했으나 검증할 그래프가 존재하지 않아 검증을 시도할 수 없었다. 이 유형을 유발한 8개 범주 또한 모두 RML 매핑 엔진이 그래프 생성에 실패해야 테스트를 통과한 것으로 판정되는 테스트 케이스들로 이루어져 있다. 이 범주의 RML 문서는 구문 오류는 없으나 의도적으로 데이터 파일에 존재하지 않는 컬럼 혹은 속성을 참조하게 함으로써 RML 매핑 엔진이 그래프 생성에 실패하도록 작성되었다. 제안 시스템은 데이터 분석 단계에서 존재하지 않는 컬럼으로의 접근이 발생하면 이 단계에서 추출할 메타데이터 없이 RML 문서만으로 스키마를 생성하도록 구현됐기 때문에 스키마 생성에 성공할 수 있었다.

유형 3 범주 테스트에서는 제안 시스템이 스키마 생성까지는 성공했으나 RDF 파일 내 타켓 노드를 성공적으로 검증하지 못했다. 이 유형을 유발한 6개 범주의 RML 문서는 모두 RDF 파일을 N-Quads[20] 포맷으로 출력하도록 작성되었다. N-Quads는 네 개의 항이 한 문장을 구성한다. 한 문장은 트리플 즉 주어·술어·목적어에 '그래프명' 항이 추가된 형식이다. 문장의 의미는 트리플이 '그래프명'으로 식별되는 그래프에 속함을 뜻한다. 제안된 시스템이 검증에 실패한 원인은 ShEx과 SHACL 두 언어 모두 언어 사양에서 트리플이 특정 그래프에 속해야 함을 제약할 수 있는 조건을 정의하고 있지 않기 때문이다.

정리하면 제안 시스템은 애초에 그래프 생성에 실패하도록 설계된 테스트 케이스와 그래프 스키마 언어 자체의 한계 때문에 검증에 실패할 수 밖에 없는 테스트 케이스 외에는 모든 테스트 케이스에서 정확한 스키마를 생성했다고 평가할 수 있다.

V. Conclusions

본 논문을 통해 RML 매핑으로 생성된 RDF 그래프의 스키마를 자동 생성해 주는 시스템을 제안했다. 제안 시스템이 기존 연구와 비교하여 갖는 의의는 두 가지로 요약될 수 있다. 첫째, 제안 시스템은 대표적인 그래프 스키마 언어인 SHACL과 ShEx 둘 다로 그래프 스키마를 출력한다는 점이다. 반면에 2.1절에서 소개된 RML 매핑을 포함하

여 R2RML 매핑 그리고 Direct Mapping으로 생성된 RDF 그래프에 대한 스키마를 자동 생성해 주는 기존 연구들은 각기 SHACL 혹은 ShEx 중 한 언어만으로 스키마를 출력했다. 제안 시스템은 공통 형상 모델이라고 명명한 구조물을 사용해 서로 다른 스키마 언어로의 출력이라 하더라도 단일한 방식으로 스키마를 생성할 수 있도록 하였다. 이 공통 형상 모델의 구조와 구축 절차 그리고 이 모델을 서로 다른 스키마 언어에서 사용하는 방법을 3장에서 자세히 기술하였다. 둘째, 제안 시스템은 검증기를 통한 그래프 구조 검증을 수행할 수 있는 수준의 스키마를 생성한다는 점이다. 2.2절에서 RML 매핑 기반의 기존 연구 [17]의 분석을 통해 도출한 기계로 검증 가능한 수준의 스키마가 되려면 갖추어야 할 네 가지 요구사항을 제안 시스템이 생성한 스키마가 갖추었음을 3.6절의 사례를 통해 보였다. 그리고 4장에서는 다양한 RML 매핑 규칙 조합과 다양한 포맷의 데이터를 가지고 제안 시스템이 생성한 스키마가 그래프 구조를 검증기로 정확히 검증할 수 있음을 테스트 결과로 보였다.

4장에서 사용한 테스트 케이스 세트는 다양한 매핑 규칙 조합을 제공하지만 입력 데이터가 모두 소규모다. 즉, 입력 데이터 규모에 따른 제안 시스템의 성능까지 함께 평가할 수 있는 도구는 아니었다. 따라서 본 연구는 데이터 규모에 따른 확장성 측면의 성능 평가가 이루어지지 않은 한계가 있다. 향후 연구로는 제안 시스템을 실제 대규모 도메인 데이터를 기반으로 한 RDF 그래프 구축 사례에 적용하는 과정을 통해 시스템의 확장성을 평가하고 보완해 나가는 작업을 지속하고자 한다.

REFERENCES

[1] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. De Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge Graphs," ACM Computing Surveys, Vol. 54, No. 4, pp. 1-37, May 2022. DOI: 10.1145/3447772

[2] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-Scale Knowledge Graphs: Lessons and Challenges," Communications of the ACM, Vol. 62, No. 8, pp. 36-43, August 2019. DOI: 10.1145/3331166

[3] H. Arnaout, and S. Elbassuoni, "Effective searching of RDF knowledge graphs," Journal of Web Semantics, Vol. 48, pp. 66-84, June 2018. DOI: 10.1016/j.websem.2017.12.001

[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] V. Janev, D. Graux, H. Jabeen, and E. Sallinger, "Knowledge Graphs and Big Data Processing," Springer Cham, pp. 59-72, July 2020.

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

[7] E. Prud'hommeaux, I. Boneva, J. E. L. Gayo, and G. Kellogg, Shape Expressions Language 2.1, https://shex.io/shex-semantics/index.html

[8] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda, A Direct Mapping of Relational Data to RDF, http://www.w3.org/TR/rdb-direct-mapping/

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

[10] R. B. Thapa, and M. Giese, "A Source-to-Target Constraint Rewriting for Direct Mapping," Proceedings of 20th International Semantic Web Conference, pp. 21-38, Virtual Event, October 2021. DOI: 10.1007/978-3-030-88361-4_2

[11] J. Choi, "Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by Direct Mappings," Journal of the Korea Society of Computer and Information Vol. 25, No. 10, pp. 23-34, October 2020. DOI: 10.9708/JKSCI.2020.25.10.023

[12] I. Boneva, J. Lozano, and S. Staworko, "Relational to RDF Data Exchange in Presence of a Shape Expression Schema," Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, pp. 1-16, Cali, Colombia, May 2018. DOI: 10.48550/arXiv.1804.11052

[13] J. Choi, "ShEx Schema Generator for RDF Graphs Created by Direct Mapping," Journal of the Korea Society of Computer and Information Vol. 23, No. 10, pp. 33-43, October 2018. DOI: 10.9708/JKSCI.2018.23.10.033

[14] I. Boneva, J. Lozano, and S. Staworko, "Consistency and Certain Answers in Relational to RDF Data Exchange with Shape Constraints," Proceedings of New Trends in Databases and Information Systems, pp. 97-107, Lyon, France, August 2020. DOI: 10.1007/978-3-030-54623-6_9

[15] J. Choi, "Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by R2RML Mappings," Journal of the Korea Society of Computer and Information Vol. 25, No. 8, pp. 9-21, August 2020. DOI: 10.9708/JKSCI.2020.25.08.009

[16] J. Choi, "R2RML Based ShEx Schema," Journal of the Korea Society of Computer and Information Vol. 23, No. 10, pp. 45-55, October 2018. DOI: 10.9708/JKSCI.2018.23.10.045

[17] 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: 10.1145/3460210.3493562

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

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

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

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.